

EXHIBIT D

D1

```

1 /*****
2  * This API is an enhancement of the RECOVER_API.
3  * It has been designed to
4  * support restoration of Symmetrix Connect backups as well as
5  * network
6  * backups. In addition, it has been implemented in a client/server
7  * architecture, where the restore GUI on an EDM client can browse,
8  * mark, and
9  * initiate restores from the EDM server.
10  * It uses the Restore Engine API
11  * to access, via ONC RPC,
12  * the restore functions provided on the EDM server.
13  *
14  * This API is defined to allow access to catalog information without
15  * knowing catalog structures or data types. This API should remain
16  * extendable through the use of data hiding techniques.
17  *
18  * Note:
19  * -----
20  * This API uses the philosophy that the caller is responsible for
21  * and freeing of objects. This should remain consistent throughout.
22  * allocation
23  *
24  ****/
25 #include <errno/e-errno.h>
26 #include <ebcfapi/Bucfg_Defs.h>
27
28 /* Need this for platform types */
29 #define RESTORE_API_VERSION 1.0
30
31 #define TIO_BITFLAG_NETWORK_RESTORE_NOT_POSSIBLE 0x1
32
33 /*
34  * Datatypes
35  */
36
37 /* restorable objects are private */
38 typedef void *restorableObjectPtr;
39
40 /* media objects are private */
41 typedef void *mediaObjectPtr;
42
43 /* feedback objects are private */
44 typedef void *feedbackObjectPtr;
45
46 /* EDMProgress object are private */
47 typedef void *EDMProgressPtr;
48
49 /* WIPProgress objects are private */
50 typedef void *WIPProgressPtr;
51
52 /* Notify objects are private */
53 typedef void *NotifyObjectPtr;
54
55 /* submit objects are private */
56 typedef void *submitUserObjectPtr;
57
58 /* query objects are opaque to the user */
59 typedef void *queryObjectPtr;
60
61 typedef enum {Backup_Good,
62
63 restore api.h 1
64
65 Page 1 of 36

```

```

59 1 Backup_Bad,
60 1 Backup_Expired,
61 Backup_Child_Without_Data) BackupStatus;

63 1 typedef enum {Media_Online,
64 1 Media_Offline,
65 Media_Offsite} MediaStatus;

67 1 typedef enum {Always_Overwrite,
68 1 Never_Overwrite,
69 Older_Only_Overwrite} OverwritePolicy;

71 1 typedef enum {Files_Only,
72 1 Directories_Only,
73 1 Others_Only,
74 All_File_Types} FileType;

76 1 typedef enum {restoreTransportSCSI,
77 restoreTransportNetwork} RestoreTransport;

79 1 typedef enum {EBREC_AllSizes,
80 1 EBREC_Equal,
81 1 EBREC_GreaterThan,
82 1 EBREC_GreaterOrEqual,
83 1 EBREC_LessThan,
84 EBREC_LessOrEqual} MatchType;

86 typedef char *hostname_ty;

88 typedef void *serverHandle;

89 /* pointer to CSC handle for Restore Engine
90 server */
91 #define MAX_TEMPLATE_LEN 64
92 typedef char template_name_ty [MAX_TEMPLATE_LEN];

93 typedef struct _EDMRST_submit_args
94 {
95 1 char *socketClientNm;
96 1 int clientSocketPort;
97 1 char *mapfile_env;
98 }EDMRST_submit_args;

104 typedef struct _EBREC_SearchCriteriaRec
105 {
106 1 char startDirectory[256]; /* Directory to start searching */
107 1 boolean_ty descendDirectory; /* Flag to descend into sub dirs */
108 1 char searchString[128];
109 1 boolean_ty excludeString; /* string to search for (as in find) */
110 1 FileType fileType; /* Flag to include or exclude string */
111 1 char owner[64]; /* Types of files to search for */
112 1 boolean_ty excludeOwner; /* Specific owner of files (or '\0') */
113 1 char group[64]; /* Flag to include or exclude owner */
114 1 boolean_ty excludeGroup; /* Specific group of files (or '\0') */
115 1 u_hyper sizeInBytes; /* Flag to include or exclude group */
116 1 MatchType sizeMatch; /* Specific size of files to find */
117 } /* The type of matching to do for size */

```

```
117 1      time_t      startTime;          /* First backup date to use for search */
118 1      time_t      endTime;              /* Last backup date to use for search */
119      } EBREC_SearchCriteriaRec;
121      /* Definition of bits in flags input to backup time selection functions: */
122      #define BACKUP_SELECTION_FLAG_MASK_COMPLETE 0x1
123      #define BACKUP_SELECTION_FLAG_COMPLETE_ONLY 0x1
124      #define BACKUP_SELECTION_FLAG_PARTIAL_OK 0x0
127      #define INIT_COOKIE 0
128      #define DONE_COOKIE -1
131      /*****
132      * Definitions to be replaced by Restore Engine Header at some point :
133      *****/
135      typedef enum {
136          RE_STATE_TIMEOUT = -4,
137          RE_STATE_ADMIN_QUIT = -3,
138          RE_STATE_USER_QUIT = -2,
139          RE_STATE_FAILED = -1,
140          RE_STATE_SUCCESSFUL = 0,
141          RE_STATE_STOPPED,
142          RE_STATE_RUNNING,
143          RE_STATE_STARTING,
144          RE_STATE_PREPHASE,
145          RE_STATE_POSTPHASE
146      } RERunningState ;
148      /*****
149      * Get Backup Servers:
150      *
151      * This function is provided to allow retrieval of the
152      * EDM servers which are available to perform restores from.
153      *
154      * This is a client-side only function.
155      *
156      * Parameters:
157      *     server (
158      *         0) - a pointer to a buffer allocated to receive a host name
159      *         IO) - a place holder for the position in the list of hosts
160      *
161      * NOTE:
162      *     this function will only return a single host name in the 4/99
163      *     release!
164      *****/
165      eerrno_ty EDMRST_GetBackupServers(
166          hostname_ty server, long *cookie );
168      /*****
169      * Initialize:
170      *
171      * This function initializes a restoral session. This must be called
172      * prior to any of the following functions in this API.
173      *****/
```

```
174      * Parameters:
175      *     serverName (I) - The machine name of the server to use.
176      *     svrHdl (
177      *         0) - A handle to receive a pointer to this user's client
178      *         handle for the Restore Engine connection.
179      *     timeout (
180      *         I) - The maximum number of seconds to wait for the connection
181      *         to the Restore Engine process to be completed.
182      *     rfc_mode (
183      *         I) - The mode inwhich a restore is going to be done, either
184      *         with RAW_NETWORK or PLUGIN.
185      *
186      * Return Codes:
187      *     E_SUCCESS - operation completed successfully
188      *     EP_RB_RECOVER_BAD_ARGS - If cannot access Dispatch
189      *     EP_RB_RECOVER_SERVER_FAIL - or restore engine host within
190      *                               timeout
191      *     EP_RB_RECOVER_NOMEM - If server access error
192      *     EP_RB_RECOVER_RPC_FAIL - If unable to use config file
193      *     EP_RB_RECOVER_PARSE_ERROR - If unable to find EB (
194      *                               sub)directory
195      *     EP_RB_RECOVER_FATALERR - Internal restore eng error
196      *     EP_RB_RECOVER_PERMISSION_DENIED - If cannot find user
197      *     EP_RB_RECOVER_INVALIDOP - If another request executing
198      *                               in password file
199      *     eerrno_ty EDMRST_initialize ( hostname_ty serverName,
200      *                               svrHdl,
201      *                               serverHandle,
202      *                               unsigned long timeout);
203      /*****
204      * Ping:
205      *
206      * This function allows a ping to be issued in order to keep the
207      * engine alive and running so that the engine will not time out.
208      *
209      * Parameters:
210      *     svrHdl (I) - A pointer to this user's client handle for the
211      *     Restore Engine (server) connection.
212      *****/
213      eerrno_ty EDMRST_Ping( serverHandle svrHdl );
216      /*****
217      * Finish:
218      *
219      * This function terminates a restoral session,
220      * but only during the browse and
221      * mark phase.
222      * It will be rejected if a restore is currently being executed.
223      * This routine will clean up any local memory used in the session
224      * and will
225      * disconnect from the Restore Engine. After calling this function,
226      * EDMRST_initialize MUST be called before calling any other
227      * functions in this
228      * API.
229      *
230      * Parameters:
231      *****/
```

```

228      * svrHdl (I) - A pointer to this user's client handle for the
229      * Restore Engine (server) connection.
230      *
231      * *****/
232      eerrno_ty EDMRST_Finish( serverHandle svrHdl );
233
234      /*****/
235      /* Storage Allocation/Deallocation Functions
236      */
237      /*****/
238      /*****
239      * Alloc Restorable Objects:
240      *
241      * This routine allocates space for the given number of restorable
242      * objects. The caller MUST pass an array of restorable objects.
243      *
244      * Parameters:
245      *
246      * svrHdl
247      *
248      *      I) - A pointer to this user's client handle for the
249      *          Restore Engine (server) connection. (
250      *          must be valid)
251      *
252      *      restorableObjects (O) - The array of objects to allocate
253      *      count (I) - The number of objects to allocate.
254      *
255      * Return Codes:
256      *
257      *      E_SUCCESS
258      *      EP_RB_RECOVER_BAD_ARGS
259      *      EP_RB_RECOVER_NOMEM
260      *
261      * *****/
262      eerrno_ty EDMRST_AllocRestorableObjects( serverHandle
263      restorableObjectPtr svrHdl,
264      restorableObjs,
265      count );
266
267      /*****
268      * Free Restorable Objects:
269      *
270      * This routine frees the space for the given number of restorable
271      * objects. The caller MUST pass an array of restorable objects.
272      *
273      * Parameters:
274      *
275      * svrHdl
276      *
277      *      I) - A pointer to this user's client handle for the
278      *          Restore Engine (server) connection. (
279      *          must be valid)
280      *
281      *      restorableObjects (IO) - The array of objects to free
282      *      count (I) - The number of objects to free.
283      *
284      * Return Codes:
285      *
286      *      E_SUCCESS
287      *      EP_RB_RECOVER_BAD_ARGS
288      *      EP_RB_RECOVER_INVALID
289      *
290      * *****/
291      eerrno_ty EDMRST_FreeRestorableObjects( serverHandle
292      restorableObjectPtr svrHdl,
293      restorableObjects,
294      count );
295
296      /*****
297      * CopyRestorableObject:
298      *
299      * This routine copies a restorable object allocating dynamic field.
300      */

```

```

287 *
288 * Parameters:
289 *   svrhdl
290 *       (I) - A pointer to this user's client handle for the
                Restore Engine (server) connection. (
                    must be valid)
291 *   sourceObject      (I) - The restorable object to copy from.
292 *   destinationObject (I) - The pre-allocated restorable object to copy to.
293 *
294 * *****/
295 eerrno_ty EDMRST_CopyRestorableObject( serverHandle
296                                     restorableObjectPtr
297                                     sourceObject,
298                                     restorableObjectPtr
299                                     *destinationObject );
300
301 /*****
302 * Alloc Media Objects:
303 *
304 * This routine allocates space for the given number of media
305 * objects. The caller MUST pass an array of media objects.
306 *
307 * Parameters:
308 *   svrhdl      (I) - A pointer to this user's client handle for the
                Restore Engine (server) connection. (
                    must be valid)
309 *   mediaObjects (O) - The array of objects to allocate
310 *   count        (I) - The number of objects to allocate.
311 *
312 * Return Codes:
313 *   E_SUCCESS
314 *   EP_RB_RECOVER_BAD_ARGS
315 *   EP_RB_RECOVER_NOMEM
316 * *****/
317 eerrno_ty EDMRST_AllocMediaObjects( serverHandle svrhdl,
318                                     mediaObjectPtr *mediaObjects,
319                                     const short count );
320
321 /*****
322 * Free Media Objects:
323 *
324 * This routine frees the space for the given number of media
325 * objects. The caller MUST pass an array of media objects.
326 *
327 * Parameters:
328 *   svrhdl
329 *       (I) - A pointer to this user's client handle for the
                Restore Engine (server) connection. (
                    must be valid)
330 *   mediaObjects (IO) - The array of objects to free
331 *   count        (I) - The number of objects to free.
332 *
333 * Return Codes:
334 *   E_SUCCESS
335 *   EP_RB_RECOVER_BAD_ARGS
336 *   EP_RB_RECOVER_INVALID
337 * *****/
338 eerrno_ty EDMRST_FreeMediaObjects( serverHandle svrhdl,
339                                     mediaObjectPtr *mediaObjects,
340                                     const short count );
341
342 /*****
343 * Alloc Query Object:
344 *
345 * *****/

```

```

345 * This routine allocates space for one query object.
346
347 * Parameters:
348 *   svrHdl (I) - A pointer to this user's client handle for the
349 *               Restore Engine (server) connection. (must be valid)
350 *   queryObjct (O) - Pointer to the query object
351
352 * Return Codes:
353 *   E_SUCCESS - operation completed successfully
354 *   EP_RB_RECOVER_BAD_ARGS
355 *   EP_RB_RECOVER_NOMEM
356 *   EDMRST_AllocQueryObjct( serverHandle svrHdl,
357 *   queryObjctPtr *queryObjct );
358
359 /*****
360 * Free Query Object:
361 *
362 * This routine frees the space for one query object.
363
364 * Parameters:
365 *   svrHdl (I) - A pointer to this user's client handle for the
366 *               Restore Engine (server) connection. (
367 *               must be valid)
368 *   queryObjct (
369 *   IO) - Ptr to the query object to free -- will be set to NULL
370
371 * Return Codes:
372 *   E_SUCCESS - operation completed successfully
373 *   EP_RB_RECOVER_BAD_ARGS
374 *   EP_RB_RECOVER_INVALID
375 *   EDMRST_FreeQueryObjct( serverHandle svrHdl,
376 *   queryObjctPtr *queryObjct );
377
378 /*****
379 * Alloc Feedback Object:
380 *
381 * This routine allocates space for one feedback object.
382
383 * Parameters:
384 *   svrHdl (I) - A pointer to this user's client handle for the
385 *               Restore Engine (server) connection. (Not used)
386 *   feedbackObjct (O) - The pointer to the allocated object
387
388 * Return Codes:
389 *   E_SUCCESS - operation completed successfully
390 *   EP_RB_RECOVER_BAD_ARGS
391 *   EP_RB_RECOVER_NOMEM
392 *   EDMRST_AllocFeedbackObjct( serverHandle svrHdl,
393 *   feedbackObjctPtr
394 *   *feedbackObjct );
395
396 /*****
397 * Free Feedback Object:
398 *
399 * This routine frees the space for one feedback object.
400
401 * Parameters:
402 *   svrHdl (I) - A pointer to this user's client handle for the
403 *               Restore Engine (server) connection. (
404 *               Not used)
405 *   feedbackObjctPtr (
406 *   IO) - The feedback object to free -- will be set to NULL

```

```

405 *
406 * Return Codes:
407 *     E_SUCCESS - operation completed successfully
408 *     EP_RB_RECOVER_BAD_ARGS
409 *     EP_RB_RECOVER_NOMEM
410 *     *****
411 *     eerrno_t EDMRST_FreezeFeedbackObject( serverHandle svrHdl,
412 *                                           feedbackObjectPtr
413 *                                           *feedbackObject );
414 *
415 * *****
416 * EDMRST_GetFeedbackStatus
417 *
418 * Function Description:
419 * Returns the pointer to the string that contains the text
420 * for the provided enum.
421 *
422 * Parameters:
423 *     status (
424 *         I) This is the enum that is returned from a GetFeedbackObject
425 *         call to the restore engine
426 *
427 * Return:
428 * On success: a pointer to a NULL-terminated string which is the
429 * text for the provided enum;
430 * On failure: a NULL pointer
431 * *****
432 * char * EDMRST_GetFeedbackStatus(RunningState status);
433 *
434 * *****
435 * The following functions obtain configuration information */
436 *
437 * from the eb.cfg database */
438 *
439 * *****
440 * Get Source Hosts:
441 *
442 * This function is provided to allow retrieval of the
443 * hosts which are restorable by a given user.
444 *
445 * Goal:
446 * For a host to be restorable there must have been at least one
447 * successful backup.
448 *
449 * The cookie must be initialize to INIT_COOKIE on the first call to
450 * this
451 * routine.
452 * This routine will update the cookie to allow retrieval of more
453 * objects if there is more than "maxEntries". The cookie will be
454 * returned as DONE_COOKIE when there are no more to retrieve.
455 *
456 * Parameters:
457 *     svrHdl
458 *         ( I) - A pointer to this user's client handle for the
459 *         Restore Engine (server) connection.
460 *     hostname
461 *         ( I) - If NULL, its a no-op. Otherwise, the list of
462 *         recoverable hosts will be filtered based on
463 *         the value of "hostname".
464 *     maxEntries
465 *         ( I) - the maximum number of hosts to return
466 *     hosts
467 *         ( O) - a pre-allocated array to return the hosts in
468 *         numberEntries (
469 *             O) - the real number of hosts returned in the array
470 *             ( IO) - a place holder for the list position
471 *     cookie
472 *         ( IO) - a place holder for the list position
473 *
474 * *****
475 * restore api.h B
476 *
477 * Page 8 of 36

```

```

464 eerrno_ty EDMRST_GetSourceHosts( serverHandle svrHdl,
465                                     const char *hostname,
466                                     const short maxEntries,
467                                     hostname_ty hosts,
468                                     short *numberEntries,
469                                     long *cookie );
470
471 /*****
472 * Get Destination Hosts:
473 *
474 * This function is provided to allow retrieval of the
475 * hosts which are allowable destinations for the source host
476 * by a given user.
477 *
478 * The cookie must be initialize to INIT_COOKIE on the first call to
479 * this routine.
480 *
481 * This routine will update the cookie to allow retrieval of more
482 * objects if there is more than "maxEntries". The cookie will be
483 * returned as DONE_COOKIE when there are no more to retrieve.
484 *
485 * Parameters:
486 *   svrHdl
487 *       I) - A pointer to this user's client handle for the
488 *           Restore Engine (server) connection.
489 *   maxEntries (I) - the maximum number of hosts to return
490 *   hosts (O) - a pre-allocated array to return the hosts in
491 *       numberEntries (
492 *           O) - the real number of hosts returned in the array
493 *           (IO) - a place holder for the list position
494 *   cookie
495 *       (IO) - a place holder for the list position
496 *
497 *****/
498
499 /*****
500 * Get Top Level Objects (formerly Get Work Items)
501 *
502 * This function is provided to allow retrieval of the work items (
503 *   network backups) and work item sets (
504 *       for Symmetrix Connect backups),
505 *   which are restorable for the given client.
506 *
507 * It is a GOAL of this routine to return all objects ever backed
508 * up successfully. Currently, though, it only looks in the config
509 * file for 'top level objects' of the given client.
510 *
511 * The cookie must be initialize to INIT_COOKIE on the first call to
512 * this routine.
513 *
514 * This routine will update the cookie to allow retrieval of more
515 * objects if there is more than "maxEntries". The cookie will be
516 * returned as DONE_COOKIE when there are no more to retrieve.
517 *
518 * Parameters:
519 *   svrHdl
520 *       I) - A pointer to this user's client handle for the
521 *           Restore Engine (server) connection.
522 *   sourceHost (
523 *       I) - A pointer to this user's client handle for the
524 *           Restore Engine (server) connection.
525 *   maxEntries (I) - the maximum number of objects to return
526 *   topLevelObjs (O) - a pre-allocated array to return the objects in
527 *       numberEntries (
528 *           O) - the real number of objects returned in the array
529 *           (IO) - a place holder for the list position
530 *   cookie
531 *       (IO) - a place holder for the list position
532 *
533 *****/
534
535 /*****
536 * Get All Top Level Objects (formerly Get Work Items)
537 *
538 * This function is provided to allow retrieval of the work items (
539 *   network backups) and work item sets (
540 *       for Symmetrix Connect backups),
541 *   which are restorable for the given client.
542 *
543 * It is a GOAL of this routine to return all objects ever backed
544 * up successfully. Currently, though, it only looks in the config
545 * file for 'top level objects' of the given client.
546 *
547 * The cookie must be initialize to INIT_COOKIE on the first call to
548 * this routine.
549 *
550 * This routine will update the cookie to allow retrieval of more
551 * objects if there is more than "maxEntries". The cookie will be
552 * returned as DONE_COOKIE when there are no more to retrieve.
553 *
554 * Parameters:
555 *   svrHdl
556 *       I) - A pointer to this user's client handle for the
557 *           Restore Engine (server) connection.
558 *   sourceHost (
559 *       I) - the name of the host whose backups are being restored
560 *       maxEntries (I) - the maximum number of objects to return
561 *       topLevelObjs (O) - a pre-allocated array to return the objects in
562 *       numberEntries (
563 *           O) - the real number of objects returned in the array
564 *           (IO) - a place holder for the list position
565 *   cookie
566 *       (IO) - a place holder for the list position
567 *
568 *****/
569
570 /*****
571 * Get Top Level Object Templates:
572 *
573 * This function is provided to allow retrieval of the
574 * templates to which a work item (top level object) belongs.
575 *****/

```

```

520 I) - the name of the host whose backups are being restored
521 * maxEntries (I) - the maximum number of objects to return
522 * topLevelObjs (
523 *   O) - a pre-allocated array to return the objects in
524 *   numberEntries (
525 *       O) - the real number of objects returned in the array
526 *       cookie (IO) - a place holder for the list position
527 *
528 *****/
529
530 /*****
531 * EDMRST_GetTopLevelObjects( serverHandle svrHdl,
532 *   const char *sourceHost,
533 *   const short maxEntries,
534 *   restorableObjectPtr *workItems,
535 *   short
536 *   long *numberEntries,
537 *   *cookie );
538 *
539 *****/
540
541 /*****
542 * Get All Top Level Objects (formerly Get Work Items)
543 *
544 * This function is provided to allow retrieval of the work items (
545 *   network backups) and work item sets (
546 *       for Symmetrix Connect backups),
547 *   which are restorable for the given client.
548 *
549 * It is a GOAL of this routine to return all objects ever backed
550 * up successfully. Currently, though, it only looks in the config
551 * file for 'top level objects' of the given client.
552 *
553 * The cookie must be initialize to INIT_COOKIE on the first call to
554 * this routine.
555 *
556 * This routine will update the cookie to allow retrieval of more
557 * objects if there is more than "maxEntries". The cookie will be
558 * returned as DONE_COOKIE when there are no more to retrieve.
559 *
560 * Parameters:
561 *   svrHdl
562 *       I) - A pointer to this user's client handle for the
563 *           Restore Engine (server) connection.
564 *   sourceHost (
565 *       I) - the name of the host whose backups are being restored
566 *       maxEntries (I) - the maximum number of objects to return
567 *       topLevelObjs (O) - a pre-allocated array to return the objects in
568 *       numberEntries (
569 *           O) - the real number of objects returned in the array
570 *           (IO) - a place holder for the list position
571 *   cookie
572 *       (IO) - a place holder for the list position
573 *
574 *****/
575
576 /*****
577 * Get Top Level Object Templates:
578 *
579 * This function is provided to allow retrieval of the
580 * templates to which a work item (top level object) belongs.
581 *****/

```

```

573 * This routine is provided in the event that the goal of the
574 * work-item time routines to include all templates cannot be met.
575 *
576 * The cookie must be initialize to INIT_COOKIE on the first call to
577 * routine.
578 * This routine will update the cookie to allow retrieval of more
579 * objects if there is more than "maxEntries". The cookie will be
580 * returned as DONE_COOKIE when there are no more to retrieve.
581 *
582 * Parameters:
583 *   svrHdl      (I) - A pointer to this user's client handle for the
584 *                 Restore Engine (server) connection.
585 *   topLevelObj (I) - the top level object in question
586 *   maxEntries  (I) - the maximum number of templates to return
587 *   templates   (O) - a pre-allocated array to return the templates in
588 *   numberEntries (O) - the real number of templates returned in the array
589 *   cookie      (IO) - a place holder for the list position
590 *
591 * *****
592 * eerrno_ty EDMRST_GetTopLevelTemplates( serverHandle svrHdl,
593 *                                       const restoreableObjectPtr
594 *                                       topLevelObj,
595 *                                       const short maxEntries,
596 *                                       template_name_ty *templates,
597 *                                       short *numberEntries,
598 *                                       long *cookie );
599 *
600 * *****
601 * Does Alternate Exist:
602 *
603 * This routine determines if an alternate trailset exists for the
604 * given
605 * template.
606 *
607 * Parameters:
608 *   svrHdl      (I) - A pointer to this user's client handle for the
609 *                 Restore Engine (server) connection.
610 *   topLevelObj (I) - the top level object in question
611 *   template    (I) - The name of the template to look for
612 *   exists      (O) - Return flag for whether or not the alternate exists
613 *
614 *   svrHdl      (I) - A pointer to this user's client handle for the
615 *                 Restore Engine (server) connection.
616 *   topLevelObj (I) - the top level object in question
617 *   template    (I) - The name of the template to look for
618 *   exists      (O) - Return flag for whether or not the alternate exists
619 *
620 * *****
621 * eerrno_ty EDMRST_DoesAlternateExist( serverHandle svrHdl,
622 *                                       const restoreableObjectPtr
623 *                                       topLevelObj,
624 *                                       template_name_ty
625 *                                       template_name,
626 *                                       boolean_ty
627 *                                       *exists );
628 *
629 * *****
630 * Set Top Level (Formerly Work Item) Template:
631 *
632 * This routine sets the template to be used by the specified top
633 * level

```

```

625 * object (work item, or CBO) and specifies whether or not to use the
626 * alternate trailset.
627 *
628 * Parameters:
629 *   svrHdl      (I) - A pointer to this user's client handle for the
630 *                 Restore Engine (server) connection.
631 *   workItem    (I) - The top level object to update
632 *   template    (I) - The name of the template to use
633 *   alternate    (I) - Flag whether or not to use the alternate trailset
634 *
635 * *****
636 * eerrno_ty EDMRST_SetTopLevelTemplate( serverHandle svrHdl,
637 *                                       const restoreableObjectPtr workItem,
638 *                                       const template_name_ty
639 *                                       template_name,
640 *                                       boolean_ty
641 *                                       alternate
642 *                                       );
643 *
644 * *****
645 * Backup Configuration Query Functions
646 *
647 * Backup Time Selection Functions
648 *
649 * Get Current Template:
650 *
651 * This routine returns the name of the template that is used by
652 * the currently selected top level object (
653 * work item) and the flag
654 * on whether or not the alternate trail is being used.
655 *
656 * Parameters:
657 *   svrHdl      (I) - A pointer to this user's client handle for the
658 *                 Restore Engine (server) connection.
659 *   template    (O) - The name of the current template
660 *   alternate    (O) - Flag whether or not the alternate trailset is being used.
661 *
662 * *****
663 * eerrno_ty EDMRST_GetCurrentTemplate( serverHandle svrHdl,
664 *                                       template_name_ty template_name,
665 *                                       boolean_ty
666 *                                       alternate );
667 *
668 * *****
669 * GetCurrentBackupTime:
670 *
671 * This routine returns the time of the backup as selected for the
672 * work item.
673 *
674 * Parameters:
675 *   svrHdl      (I) - A pointer to this user's client handle for the
676 *                 Restore Engine (server) connection.
677 *   time        (O) - The time the backup for the work-item was run
678 *
679 * *****
680 * eerrno_ty EDMRST_GetCurrentBackupTime( serverHandle svrHdl,
681 *                                       time_t
682 *                                       *time );
683 *
684 * *****
685 * Backup Time Selection Functions
686 *
687 * Backup Time Selection Functions
688 *
689 * Backup Time Selection Functions
690 *
691 * Backup Time Selection Functions
692 *
693 * Backup Time Selection Functions
694 *
695 * Backup Time Selection Functions
696 *
697 * Backup Time Selection Functions
698 *
699 * Backup Time Selection Functions
700 *
701 * Backup Time Selection Functions
702 *
703 * Backup Time Selection Functions
704 *
705 * Backup Time Selection Functions
706 *
707 * Backup Time Selection Functions
708 *
709 * Backup Time Selection Functions
710 *
711 * Backup Time Selection Functions
712 *
713 * Backup Time Selection Functions
714 *
715 * Backup Time Selection Functions
716 *
717 * Backup Time Selection Functions
718 *
719 * Backup Time Selection Functions
720 *
721 * Backup Time Selection Functions
722 *
723 * Backup Time Selection Functions
724 *
725 * Backup Time Selection Functions
726 *
727 * Backup Time Selection Functions
728 *
729 * Backup Time Selection Functions
730 *
731 * Backup Time Selection Functions
732 *
733 * Backup Time Selection Functions
734 *
735 * Backup Time Selection Functions
736 *
737 * Backup Time Selection Functions
738 *
739 * Backup Time Selection Functions
740 *
741 * Backup Time Selection Functions
742 *
743 * Backup Time Selection Functions
744 *
745 * Backup Time Selection Functions
746 *
747 * Backup Time Selection Functions
748 *
749 * Backup Time Selection Functions
750 *
751 * Backup Time Selection Functions
752 *
753 * Backup Time Selection Functions
754 *
755 * Backup Time Selection Functions
756 *
757 * Backup Time Selection Functions
758 *
759 * Backup Time Selection Functions
760 *
761 * Backup Time Selection Functions
762 *
763 * Backup Time Selection Functions
764 *
765 * Backup Time Selection Functions
766 *
767 * Backup Time Selection Functions
768 *
769 * Backup Time Selection Functions
770 *
771 * Backup Time Selection Functions
772 *
773 * Backup Time Selection Functions
774 *
775 * Backup Time Selection Functions
776 *
777 * Backup Time Selection Functions
778 *
779 * Backup Time Selection Functions
780 *
781 * Backup Time Selection Functions
782 *
783 * Backup Time Selection Functions
784 *
785 * Backup Time Selection Functions
786 *
787 * Backup Time Selection Functions
788 *
789 * Backup Time Selection Functions
790 *
791 * Backup Time Selection Functions
792 *
793 * Backup Time Selection Functions
794 *
795 * Backup Time Selection Functions
796 *
797 * Backup Time Selection Functions
798 *
799 * Backup Time Selection Functions
800 *
801 * Backup Time Selection Functions
802 *
803 * Backup Time Selection Functions
804 *
805 * Backup Time Selection Functions
806 *
807 * Backup Time Selection Functions
808 *
809 * Backup Time Selection Functions
810 *
811 * Backup Time Selection Functions
812 *
813 * Backup Time Selection Functions
814 *
815 * Backup Time Selection Functions
816 *
817 * Backup Time Selection Functions
818 *
819 * Backup Time Selection Functions
820 *
821 * Backup Time Selection Functions
822 *
823 * Backup Time Selection Functions
824 *
825 * Backup Time Selection Functions
826 *
827 * Backup Time Selection Functions
828 *
829 * Backup Time Selection Functions
830 *
831 * Backup Time Selection Functions
832 *
833 * Backup Time Selection Functions
834 *
835 * Backup Time Selection Functions
836 *
837 * Backup Time Selection Functions
838 *
839 * Backup Time Selection Functions
840 *
841 * Backup Time Selection Functions
842 *
843 * Backup Time Selection Functions
844 *
845 * Backup Time Selection Functions
846 *
847 * Backup Time Selection Functions
848 *
849 * Backup Time Selection Functions
850 *
851 * Backup Time Selection Functions
852 *
853 * Backup Time Selection Functions
854 *
855 * Backup Time Selection Functions
856 *
857 * Backup Time Selection Functions
858 *
859 * Backup Time Selection Functions
860 *
861 * Backup Time Selection Functions
862 *
863 * Backup Time Selection Functions
864 *
865 * Backup Time Selection Functions
866 *
867 * Backup Time Selection Functions
868 *
869 * Backup Time Selection Functions
870 *
871 * Backup Time Selection Functions
872 *
873 * Backup Time Selection Functions
874 *
875 * Backup Time Selection Functions
876 *
877 * Backup Time Selection Functions
878 *
879 * Backup Time Selection Functions
880 *
881 * Backup Time Selection Functions
882 *
883 * Backup Time Selection Functions
884 *
885 * Backup Time Selection Functions
886 *
887 * Backup Time Selection Functions
888 *
889 * Backup Time Selection Functions
890 *
891 * Backup Time Selection Functions
892 *
893 * Backup Time Selection Functions
894 *
895 * Backup Time Selection Functions
896 *
897 * Backup Time Selection Functions
898 *
899 * Backup Time Selection Functions
900 *
901 * Backup Time Selection Functions
902 *
903 * Backup Time Selection Functions
904 *
905 * Backup Time Selection Functions
906 *
907 * Backup Time Selection Functions
908 *
909 * Backup Time Selection Functions
910 *
911 * Backup Time Selection Functions
912 *
913 * Backup Time Selection Functions
914 *
915 * Backup Time Selection Functions
916 *
917 * Backup Time Selection Functions
918 *
919 * Backup Time Selection Functions
920 *
921 * Backup Time Selection Functions
922 *
923 * Backup Time Selection Functions
924 *
925 * Backup Time Selection Functions
926 *
927 * Backup Time Selection Functions
928 *
929 * Backup Time Selection Functions
930 *
931 * Backup Time Selection Functions
932 *
933 * Backup Time Selection Functions
934 *
935 * Backup Time Selection Functions
936 *
937 * Backup Time Selection Functions
938 *
939 * Backup Time Selection Functions
940 *
941 * Backup Time Selection Functions
942 *
943 * Backup Time Selection Functions
944 *
945 * Backup Time Selection Functions
946 *
947 * Backup Time Selection Functions
948 *
949 * Backup Time Selection Functions
950 *
951 * Backup Time Selection Functions
952 *
953 * Backup Time Selection Functions
954 *
955 * Backup Time Selection Functions
956 *
957 * Backup Time Selection Functions
958 *
959 * Backup Time Selection Functions
960 *
961 * Backup Time Selection Functions
962 *
963 * Backup Time Selection Functions
964 *
965 * Backup Time Selection Functions
966 *
967 * Backup Time Selection Functions
968 *
969 * Backup Time Selection Functions
970 *
971 * Backup Time Selection Functions
972 *
973 * Backup Time Selection Functions
974 *
975 * Backup Time Selection Functions
976 *
977 * Backup Time Selection Functions
978 *
979 * Backup Time Selection Functions
980 *
981 * Backup Time Selection Functions
982 *
983 * Backup Time Selection Functions
984 *
985 * Backup Time Selection Functions
986 *
987 * Backup Time Selection Functions
988 *
989 * Backup Time Selection Functions
990 *
991 * Backup Time Selection Functions
992 *
993 * Backup Time Selection Functions
994 *
995 * Backup Time Selection Functions
996 *
997 * Backup Time Selection Functions
998 *
999 * Backup Time Selection Functions
1000 *

```

```

655 /*****
656 * SetPrevBackup
657 *
658 * This routine sets the recover environment to the previous backup
659 * of the currently selected work item
660 *
661 * Goal (?) :
662 * This includes both primary and alternate trailsets for all
663 * templates for which this work-item has backups.
664 *
665 * Parameters:
666 * svrhd1 (I) - A pointer to this user's client handle for the
667 * Restore Engine (server) connection.
668 *
669 * flags (
670 * I) - Selection Flags: e.g., Complete backups only/partial ok
671 *
672 * *****/
673 eerrno_ty EDMRST_SetPrevBackup( serverHandle svrhd1,
674                                u_long flags );
675
676 /*****
677 * SetNextBackup
678 *
679 * This routine sets the recover environment to the next backup
680 * of the specified work item.
681 *
682 * Goal:
683 * This includes both primary and alternate trailsets for all
684 * templates for which this work-item has backups.
685 *
686 * Parameters:
687 * svrhd1 (I) - A pointer to this user's client handle for the
688 * Restore Engine (server) connection.
689 *
690 * flags (
691 * I) - Selection Flags: e.g., Complete backups only/partial ok
692 *
693 * *****/
694 eerrno_ty EDMRST_SetNextBackup( serverHandle svrhd1,
695                                u_long flags );
696
697 /*****
698 * SetFirstBackup
699 *
700 * This routine sets the recover environment to the first backup
701 * kept for the specified work item.
702 *
703 * Goal:
704 * This includes both primary and alternate trailsets for all
705 * templates for which this work-item has backups.
706 *
707 * Parameters:
708 * svrhd1 (I) - A pointer to this user's client handle for the
709 * Restore Engine (server) connection.
710 *
711 * flags (
712 * I) - Selection Flags: e.g., Complete backups only/partial ok
713 *
714 * *****/
715 eerrno_ty EDMRST_SetFirstBackup( serverHandle svrhd1,
716                                  u_long flags );
717
718 /*****
719 * SetMostRecentBackup
720 *
721 * This routine sets the recover environment to the most recent
722 * backup
723 * *****/
724 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
725                                         u_long flags );
726
727 /*****
728 * SetMostRecentBackup
729 *
730 * This routine sets the recover environment to the most recent
731 * backup
732 * *****/
733 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
734                                         u_long flags );
735
736 /*****
737 * SetMostRecentBackup
738 *
739 * This routine sets the recover environment to the most recent
740 * backup
741 * *****/
742 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
743                                         u_long flags );
744
745 /*****
746 * SetMostRecentBackup
747 *
748 * This routine sets the recover environment to the most recent
749 * backup
750 * *****/
751 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
752                                         u_long flags );
753
754 /*****
755 * SetMostRecentBackup
756 *
757 * This routine sets the recover environment to the most recent
758 * backup
759 * *****/
760 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
761                                         u_long flags );
762
763 /*****
764 * SetMostRecentBackup
765 *
766 * This routine sets the recover environment to the most recent
767 * backup
768 * *****/
769 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
770                                         u_long flags );
771
772 /*****
773 * SetMostRecentBackup
774 *
775 * This routine sets the recover environment to the most recent
776 * backup
777 * *****/
778 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
779                                         u_long flags );
780
781 /*****
782 * SetMostRecentBackup
783 *
784 * This routine sets the recover environment to the most recent
785 * backup
786 * *****/
787 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
788                                         u_long flags );
789
790 /*****
791 * SetMostRecentBackup
792 *
793 * This routine sets the recover environment to the most recent
794 * backup
795 * *****/
796 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
797                                         u_long flags );
798
799 /*****
800 * SetMostRecentBackup
801 *
802 * This routine sets the recover environment to the most recent
803 * backup
804 * *****/
805 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
806                                         u_long flags );
807
808 /*****
809 * SetMostRecentBackup
810 *
811 * This routine sets the recover environment to the most recent
812 * backup
813 * *****/
814 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
815                                         u_long flags );
816
817 /*****
818 * SetMostRecentBackup
819 *
820 * This routine sets the recover environment to the most recent
821 * backup
822 * *****/
823 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
824                                         u_long flags );
825
826 /*****
827 * SetMostRecentBackup
828 *
829 * This routine sets the recover environment to the most recent
830 * backup
831 * *****/
832 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
833                                         u_long flags );
834
835 /*****
836 * SetMostRecentBackup
837 *
838 * This routine sets the recover environment to the most recent
839 * backup
840 * *****/
841 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
842                                         u_long flags );
843
844 /*****
845 * SetMostRecentBackup
846 *
847 * This routine sets the recover environment to the most recent
848 * backup
849 * *****/
850 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
851                                         u_long flags );
852
853 /*****
854 * SetMostRecentBackup
855 *
856 * This routine sets the recover environment to the most recent
857 * backup
858 * *****/
859 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
860                                         u_long flags );
861
862 /*****
863 * SetMostRecentBackup
864 *
865 * This routine sets the recover environment to the most recent
866 * backup
867 * *****/
868 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
869                                         u_long flags );
870
871 /*****
872 * SetMostRecentBackup
873 *
874 * This routine sets the recover environment to the most recent
875 * backup
876 * *****/
877 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
878                                         u_long flags );
879
880 /*****
881 * SetMostRecentBackup
882 *
883 * This routine sets the recover environment to the most recent
884 * backup
885 * *****/
886 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
887                                         u_long flags );
888
889 /*****
890 * SetMostRecentBackup
891 *
892 * This routine sets the recover environment to the most recent
893 * backup
894 * *****/
895 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
896                                         u_long flags );
897
898 /*****
899 * SetMostRecentBackup
900 *
901 * This routine sets the recover environment to the most recent
902 * backup
903 * *****/
904 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
905                                         u_long flags );
906
907 /*****
908 * SetMostRecentBackup
909 *
910 * This routine sets the recover environment to the most recent
911 * backup
912 * *****/
913 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
914                                         u_long flags );
915
916 /*****
917 * SetMostRecentBackup
918 *
919 * This routine sets the recover environment to the most recent
920 * backup
921 * *****/
922 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
923                                         u_long flags );
924
925 /*****
926 * SetMostRecentBackup
927 *
928 * This routine sets the recover environment to the most recent
929 * backup
929 * *****/
930 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
931                                         u_long flags );
932
933 /*****
934 * SetMostRecentBackup
935 *
936 * This routine sets the recover environment to the most recent
937 * backup
938 * *****/
939 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
940                                         u_long flags );
941
942 /*****
943 * SetMostRecentBackup
944 *
945 * This routine sets the recover environment to the most recent
946 * backup
947 * *****/
948 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
949                                         u_long flags );
950
951 /*****
952 * SetMostRecentBackup
953 *
954 * This routine sets the recover environment to the most recent
955 * backup
956 * *****/
957 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
958                                         u_long flags );
959
960 /*****
961 * SetMostRecentBackup
962 *
963 * This routine sets the recover environment to the most recent
964 * backup
965 * *****/
966 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
967                                         u_long flags );
968
969 /*****
970 * SetMostRecentBackup
971 *
972 * This routine sets the recover environment to the most recent
973 * backup
974 * *****/
975 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
976                                         u_long flags );
977
978 /*****
979 * SetMostRecentBackup
980 *
981 * This routine sets the recover environment to the most recent
982 * backup
983 * *****/
984 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
985                                         u_long flags );
986
987 /*****
988 * SetMostRecentBackup
989 *
990 * This routine sets the recover environment to the most recent
991 * backup
992 * *****/
993 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
994                                         u_long flags );
995
996 /*****
997 * SetMostRecentBackup
998 *
999 * This routine sets the recover environment to the most recent
1000 * backup
1001 * *****/
1002 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
1003                                         u_long flags );
1004
1005 /*****
1006 * SetMostRecentBackup
1007 *
1008 * This routine sets the recover environment to the most recent
1009 * backup
1010 * *****/
1011 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
1012                                         u_long flags );
1013
1014 /*****
1015 * SetMostRecentBackup
1016 *
1017 * This routine sets the recover environment to the most recent
1018 * backup
1019 * *****/
1020 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
1021                                         u_long flags );
1022
1023 /*****
1024 * SetMostRecentBackup
1025 *
1026 * This routine sets the recover environment to the most recent
1027 * backup
1028 * *****/
1029 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
1030                                         u_long flags );
1031
1032 /*****
1033 * SetMostRecentBackup
1034 *
1035 * This routine sets the recover environment to the most recent
1036 * backup
1037 * *****/
1038 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
1039                                         u_long flags );
1040
1041 /*****
1042 * SetMostRecentBackup
1043 *
1044 * This routine sets the recover environment to the most recent
1045 * backup
1046 * *****/
1047 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
1048                                         u_long flags );
1049
1050 /*****
1051 * SetMostRecentBackup
1052 *
1053 * This routine sets the recover environment to the most recent
1054 * backup
1055 * *****/
1056 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
1057                                         u_long flags );
1058
1059 /*****
1060 * SetMostRecentBackup
1061 *
1062 * This routine sets the recover environment to the most recent
1063 * backup
1064 * *****/
1065 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
1066                                         u_long flags );
1067
1068 /*****
1069 * SetMostRecentBackup
1070 *
1071 * This routine sets the recover environment to the most recent
1072 * backup
1073 * *****/
1074 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1,
1075                                         u_long flags );
1076
1077 /*****
1078 * SetMostRecentBackup
1079 *
1080 * This routine sets the recover environment to the most recent
1081 * backup
1082 * *****/
1083 eerrno_ty
```

```

746 * of the specified work item.
747
748 * Goal:
749 * This includes both primary and alternate trailsets for all
750 * templates for which this work-item has backups.
751
752 * Parameters:
753 * svrhd1 (I) - A pointer to this user's client handle for the
754 * Restore Engine (server) connection.
755 * flags (I) - Selection flags: e.g., Complete backups only/partial ok
756
757 *****
758 eerrno_ty EDMRST_SetMostRecentBackup( serverHandle svrhd1, flags );
759
760 /*****
761 * Get All Backup Times:
762
763 * This function is provided to allow retrieval of the times which
764 * the given work-item was backed up.
765
766 * The cookie must be initialize to INIT_COOKIE on the first call to
767 * this routine.
768
769 * This routine will update the cookie to allow retrieval of more
770 * objects if there is more than "maxEntries". The cookie will be
771 * returned as DONE_COOKIE when there are no more to retrieve.
772
773 * Parameters:
774 * svrhd1 (I) - A pointer to this user's client handle for the
775 * Restore Engine (server) connection.
776 * startTime (I) - first time which should be returned ( nothing before)
777 * endTime (I) - last time which should be returned ( nothing after)
778
779 * flags (I) - Backup constraint flags: e.g. full-only/partial-ok
780 * maxEntries (I) - the maximum number of templates to return
781 * times (O) - a pre-allocated array to return the times in
782 * numberEntries (O) - the real number of templates returned in the array
783 * cookie (IO) - a place holder for the list position
784
785 *****
786 eerrno_ty EDMRST_GetAllBackupTimes( serverHandle svrhd1,
787 const time_t startTime,
788 const time_t endTime,
789 u_long flags,
790 const short maxEntries,
791 time_t *times,
792 short *numberEntries,
793 long *cookie );
794
795 /*****
796 * Set Backup for Time
797
798 * This routine sets the recover environment to the given backup time
799
800 * Goal:
801 * This includes both primary and alternate trailsets for all
802 * templates for which this work-item has backups.
803
804 * Parameters:
805 * svrhd1 (I) - A pointer to this user's client handle for the
806 * restore.apl.h 14

```

```
804 * Restore Engine (server) connection.
805 * time (I) - The time desired
806 * flags (I) - Selection Flags: e.g., Complete backups only/partial ok
807 *
808 * *****
809 * eerrno_ty EDMRST_SetBackupForTime( serverHandle svrHdl,
810 *     const time_t time,
811 *     u_long flags );
812 *
813 * *****
814 * Get Restorable Objects:
815 *
816 * *
817 * These functions are provided to allow retrieval of the
818 * child objects which are restorable given the parent object. The
819 * caller specifies the parent object and whether or not
820 * to include bad files.
821 *
822 * The cookie must be initialize to INIT_COOKIE on the first call to
823 * this routine.
824 * This routine will update the cookie to allow retrieval of more
825 * objects if there is more than "maxEntries". The cookie will be
826 * returned as DONE_COOKIE when there are no more to retrieve.
827 *
828 * Parameters:
829 *     svrHdl (I) - A pointer to this user's client handle for the
830 *         Restore Engine (server) connection.
831 *     parentObject (I) - the parent object
832 *     allowBadFiles (I) - flag whether or not to include bad files
833 *     maxEntries (I) - the maximum number of objects to return
834 *     objects (O) - a pre-allocated array to return the objects in
835 *         numberEntries (O) - the real number of objects returned in the array
836 *     cookie (IO) - a place holder for the list position
837 *
838 * *****
839 * eerrno_ty EDMRST_GetRestorableObjects( serverHandle svrHdl,
840 *     const restorableObjectPtr parentObject,
841 *     const boolean_ty allowBadFiles,
842 *     const long maxEntries,
843 *     long restorableObjectPtr *objects,
844 *     long *numberEntries,
845 *     long *cookie );
846 *
847 * *****
848 * Restorable Object Access Routines:
849 *
850 * *
851 * These routines retrieve information pertinent to a given
852 * restorable object.
853 *
854 * Notes:
855 * *
856 * GetObjectSize - returns the size of the individual object
857 *
858 * Parameters:
859 * *
860 *     svrHdl (I) - A pointer to this user's client handle for the
861 *         Restore Engine (server) connection.
862 *     restorable_obj (I) - The restorable object
```

```
860 * thisObject (I) - The restoral object
861 *
862 * *****
863 * const char *EDMRST_GetObjectBaseName( serverHandle svrHdl,
864 *     restorableObjectPtr thisObject );
865 *
866 * const char *EDMRST_GetObjectFullName( serverHandle svrHdl,
867 *     restorableObjectPtr thisObject );
868 *
869 * boolean_ty EDMRST_IsObjectTopLevel( serverHandle svrHdl,
870 *     restorableObjectPtr thisObject );
871 *
872 * boolean_ty EDMRST_IsObjectContainer( serverHandle svrHdl,
873 *     restorableObjectPtr thisObject );
874 *
875 * boolean_ty EDMRST_IsObjectLeaf( serverHandle svrHdl,
876 *     restorableObjectPtr thisObject );
877 *
878 * const char *EDMRST_GetObjectTypeString( serverHandle svrHdl,
879 *     restorableObjectPtr thisObject );
880 *
881 * u_long EDMRST_GetObjectPermissions( serverHandle svrHdl,
882 *     restorableObjectPtr thisObject );
883 *
884 * const char *EDMRST_GetObjectNameString( serverHandle svrHdl,
885 *     restorableObjectPtr thisObject );
886 *
887 * const char *EDMRST_GetObjectGroupString( serverHandle svrHdl,
888 *     restorableObjectPtr thisObject );
889 *
890 * time_t EDMRST_GetObjectModDate( serverHandle svrHdl,
891 *     restorableObjectPtr thisObject );
892 *
893 * u_hyper EDMRST_GetObjectSize( serverHandle svrHdl,
894 *     restorableObjectPtr thisObject );
895 *
896 * BackupStatus EDMRST_GetObjectStatus( serverHandle svrHdl,
897 *     restorableObjectPtr thisObject );
898 *
899 * char EDMRST_GetWorkItemName( serverHandle svrHdl,
900 *     restorableObjectPtr thisObject );
901 *
902 * const char *EDMRST_GetWorkItemBIC( serverHandle svrHdl,
903 *     restorableObjectPtr thisObject );
904 *
905 * *****
906 * Is Object Markable
907 *
908 * *
909 * This routine determines if the current user is able to restore the
910 * given object.
911 *
912 * Parameters:
913 * *
914 *     svrHdl (I) - A pointer to this user's client handle for the
915 *         Restore Engine (server) connection.
916 *     thisObject (I) - The restoral object
917 *     markable (O) - TRUE/FALSE indicating markable or not
918 *
919 * *****
```

```

920 eerrno_ty EDMRST_IsObjectMarkable( serverHandle      svrHdl,
921                                     const restoreableObjectPtr thisObject,
922                                     boolean_ty      *markable );
923
924 /*****
925  * Is Object Searchable
926  *
927  * This routine determines if a top level object supports the Find
928  * function.
929  *
930  * Parameters:
931  *   svrHdl      (I) - A pointer to this user's client handle for the
932  *                   Restore Engine (server) connection.
933  *   thisObject  (I) - The restoreal object to query
934  *   searchable  (O) - TRUE/FALSE indicating searchable or not
935  *
936  * Return Codes:
937  *   E_SUCCESS      - operation completed successfully
938  *   EP_RB_RECOVER_BAD_ARGS
939  *   EP_RB_RECOVER_RPC_FAIL
940  *   EP_RB_RECOVER_SERVER_FAIL
941  *   EP_RB_RECOVER_INVALIDOP      -If another request active
942  *
943  *****/
944
945 eerrno_ty EDMRST_IsObjectSearchable( serverHandle      svrHdl,
946                                     const restoreableObjectPtr thisObject,
947                                     boolean_ty      *searchable );
948
949 /*****
950  * Get Symmetric Support
951  *
952  * This routine determines if a top level object supports restoreal
953  * thru a Symm
954  *
955  * Parameters:
956  *   svrHdl      (I) - A pointer to this user's client handle for the
957  *                   Restore Engine (server) connection.
958  *   thisObject  (I) - The restoreal object to query
959  *   symm_restorable  (O) - TRUE/FALSE indicating restorable over Symm (
960  *                           SCSI)
961  *
962  * Return Codes:
963  *   E_SUCCESS      - operation completed successfully
964  *   EP_RB_RECOVER_BAD_ARGS
965  *   EP_RB_RECOVER_RPC_FAIL
966  *   EP_RB_RECOVER_SERVER_FAIL
967  *   EP_RB_RECOVER_INVALIDOP      -If another request active
968  *
969  *****/
970
971
972
973 eerrno_ty EDMRST_GetSymmRestoreOption( serverHandle      svrHdl,
974                                     const restoreableObjectPtr thisObject,
975                                     boolean_ty      *symm_restorable );

```

```

977 /*****
978  * Get Network Support
979  *
980  * This routine determines if a top level object supports restoreal
981  * thru
982  * the network
983  *
984  * Parameters:
985  *   svrHdl      (I) - A pointer to this user's client handle for the
986  *                   Restore Engine (server) connection.
987  *   thisObject  (I) - The restoreal object to query
988  *   net_restorable  (O) - TRUE/FALSE indicating restorable over Symm (
989  *                           SCSI)
990  *
991  * Return Codes:
992  *   E_SUCCESS      - operation completed successfully
993  *   EP_RB_RECOVER_BAD_ARGS
994  *   EP_RB_RECOVER_RPC_FAIL
995  *   EP_RB_RECOVER_SERVER_FAIL
996  *   EP_RB_RECOVER_INVALIDOP      -If another request active
997  *
998  *****/
999
1000 eerrno_ty EDMRST_GetNetworkRestoreOption( serverHandle      svrHdl,
1001                                     const restoreableObjectPtr thisObject,
1002                                     boolean_ty      *net_restorable );
1003
1004 /*****
1005  * Find Routines:
1006  *
1007  * These routines allow the user to find restoreable objects.
1008  * Returned
1009  * is an array of found objects and an array of backup times
1010  * associated
1011  * with the objects. These arrays are 1-to-1. That is,
1012  * the nth object
1013  * was backed up at the nth time.
1014  *
1015  * This operation is performed asynchronously by the Restore Engine,
1016  * and the
1017  * first API function, EDMRST_FindRestoreableObjects,
1018  * is used to start the
1019  * 'find'.
1020  * EDMRST_GetFindResults is used to test for completion of the find,
1021  * and receive the results (parts of, at least) if it is done.
1022  *
1023  * EDMRST_FindRestoreableObjects Parameters:
1024  *
1025  *   svrHdl      (I) - A pointer to this user's client handle for the
1026  *                   Restore Engine (server) connection.
1027  *   searchCriteria  (I) - The criteria used for the search
1028  *
1029  *****/
1030
1031
1032
1033 eerrno_ty EDMRST_FindRestoreableObjects(
1034                                     serverHandle      svrHdl,
1035                                     EBREC_SearchCriteriaRec searchCriteria );

```

```

1028 /*****
1029  * EDMRST_GetFindResults Parameters:
1030  *
1031  *   svrHdl      (I) - a pointer to this user's client handle for the
1032  *               Restore Engine (server) connection.
1033  *   interrupt    (I) - requests cancellation of the find (if TRUE)
1034  *   maxEntries   (I) - the maximum number of found objects to return
1035  *   objects      (I) - a pre-allocated array to return the objects in
1036  *               times
1037  *               O) - a pre-allocated array to return the backup times in
1038  *               numberEntries (I) - the real number of objects returned in the array
1039  *               cookie        (IO) - a place holder for the list position
1040  *****/
1041  errno_t EDMRST_GetFindResults( serverHandle  svrHdl,
1042                                boolean_t     interrupt,
1043                                const long    maxEntries,
1044                                restoreableObjectPtr *foundObjects,
1045                                time_t        times,
1046                                long          *numberEntries,
1047                                long          *cookie );
1048
1049 /*****
1050  * MarkObject() and GetMarkResults()
1051  *
1052  *   MarkObject is passed a restoreableObject and marks files for
1053  *   recovery based on the input criteria.
1054  *   It starts an asynchronous operation
1055  *   in the Restore Engine to perform the marking, and returns.
1056  *
1057  *   GetMarkResults is called to test for completion of the mark
1058  *   operation,
1059  *   and receive results when it is done.
1060  *   It can also be used to interrupt
1061  *   the mark operation.
1062  *
1063  *   MarkObject Parameters:
1064  *
1065  *   svrHdl      (I) - A pointer to this user's client handle for the
1066  *               Restore Engine (server) connection.
1067  *   thisObject  (I) - The restoral object;
1068  *               can be a leaf object (e.g. a
1069  *               file), or a container object (
1070  *               e.g., a directory).
1071  *   time        (I) - (
1072  *               optional) the backup time to perform the mark on --
1073  *               if not specified,
1074  *               uses currently selected backup; if
1075  *               specified,
1076  *               leaves selected backup time unchanged
1077  *   allowBadfiles (I) - allows marking of files of state BADDATA.
1078  *   descend      (I) - Should mark operation descend to operate on the content
1079  *               of container objects.
1080  *****/
1081  errno_t EDMRST_MarkObject( serverHandle  svrHdl,
1082                             restoreableObjectPtr thisObject,
1083                             time_t        time,
1084                             boolean_t     allowBadfiles,
1085                             restore_apih_t

```

```

1079  boolean_t     descend );
1080
1081 /*****
1082  *
1083  *   GetMarkResults Parameters:
1084  *
1085  *   svrHdl      (I) - A pointer to this user's client handle for the
1086  *               Restore Engine (server) connection.
1087  *   interrupt    (I) - requests cancellation of the mark (if TRUE)
1088  *               WARNING: If the operation is aborted,
1089  *               the mark will be
1090  *               left in an unknown state. That is,
1091  *               any one of the
1092  *               descendants of the marked object may be
1093  *               marked or not.
1094  *               It is up to the caller to determine how to
1095  *               proceed
1096  *               afterwards.
1097  *   BadFilesCount (O) - returns the file count with BADDATA.
1098  *   PermDenyFilesCount (I)
1099  *   PermDenyFilesCount (O) -- returns the file count with permission denied.
1100  *
1101  *   fileMarked   (I) - return the total files marked after this mark occurred.
1102  *   dirMarked    (I) - return the total directories marked after this mark
1103  *   occurred.
1104  *   otherMarked  (I) - return the total "other" files marked after this mark.
1105  *
1106  *   O) - return the total "other" files marked after this mark.
1107  *****/
1108  errno_t EDMRST_GetMarkResults( serverHandle  svrHdl,
1109                                boolean_t     interrupt,
1110                                u_long        *BadFilesCount,
1111                                u_long        *PermDenyFilesCount,
1112                                u_long        *fileMarked,
1113                                u_long        *dirMarked,
1114                                u_long        *otherMarked );
1115
1116 /*****
1117  * UmarkObject() and GetUmarkResults()
1118  *
1119  *   UmarkObject operates like MarkObject,
1120  *   in that it is supported through
1121  *   two API calls -- UmarkObject and GetUmarkResults.
1122  *   Umark starts an
1123  *   asynchronous operation in the Restore Engine to perform the
1124  *   unmarking,
1125  *   and returns.
1126  *
1127  *   GetUmarkResults is called to test for completion of the unmark
1128  *   operation,
1129  *   and receive results when it is done.
1130  *   It can also be used to interrupt
1131  *   the unmark operation.
1132  *
1133  *   UmarkObject Parameters:
1134  *
1135  *   svrHdl      (I) - A pointer to this user's client handle for the
1136  *               Restore Engine (server) connection.
1137  *   thisObject  (I) - The restoral object;
1138  *               can be a leaf object (e.g. a

```

```

1127 * file), or a container object (
1128 * time (I) - ( e.g., a directory).
1129 * optional) the backup time to perform the unmark on --
1130 * if not specified,
1131 * uses currently selected backup; if
1132 * specified,
1133 * leaves selected backup time unchanged
1134 * BadFilesOnly (
1135 * I) - allows unmarking ONLY of files of state BADDATA.
1136 * descend (
1137 * I) - Should unmark operation descend to operate on the
1138 * content of container objects.
1139 * ****
1140 *
1141 * eerrno_ty EDMRST_UnmarkObject( serverHandle svrHdl,
1142 * restoreableObjectPtr thisObject,
1143 * time_t time,
1144 * boolean_ty BadFilesOnly,
1145 * boolean_ty descend );
1146 *
1147 * ****
1148 * GetUnmarkResults Parameters:
1149 *
1150 * svrHdl (
1151 * I) - A pointer to this user's client handle for the
1152 * Restore Engine (server) connection.
1153 * Interrupt (I) - requests cancellation of the unmark (if TRUE)
1154 * WARNING: If the operation is aborted,
1155 * the unmark will
1156 * be left in an unknown state.
1157 * That is,
1158 * any one of the
1159 * descendants of the unmarked object may be
1160 * marked or
1161 * not.
1162 * It is up to the caller to determine how to
1163 * proceed afterwards.
1164 * BadFilesCount (O) - returns the file count with BADDATA.
1165 * fileMarked (
1166 * O) - return the total files marked after this mark occurred.
1167 * dirMarked (
1168 * O) - return the total directories marked after this mark
1169 * occurred.
1170 * otherMarked (
1171 * O) - return the total "other" files marked after this mark.
1172 * ****
1173 * eerrno_ty EDMRST_GetUnmarkResults( serverHandle svrHdl,
1174 * boolean_ty interrupt,
1175 * u_long *BadFilesCount,
1176 * u_long *fileMarked,
1177 * u_long *dirMarked,
1178 * u_long *otherMarked );
1179 *
1180 * ****
1181 * IsObjectMarked:
1182 *
1183 * This function determines if each of a set of restorable objects
1184 * has been
1185 * marked for restoration.
1186 * It is intended to allow the user to determine the
1187 * current restore markings for the restorable objects at the same
1188 * object tree
1189 *
1190 * Fri Oct 10 14:57:43 2008 restore_aplh.21 Page 21 of 36

```

```

1175 * hierarchy level,
1176 * i.e. objects that have the same parent restorableObject.
1177 *
1178 * Parameters:
1179 * svrHdl (I) - A pointer to this user's client handle for the
1180 * Restore Engine (server) connection.
1181 * numEntries (I) - number of objects in bufPtrArray to be checked
1182 * bufPtrArray (I) - restorable objects to be checked for marks
1183 * numChecked (O) - number of objects in bufPtrArray found marked
1184 * marked (O) - array of booleans to receive individual marked(
1185 * 1) /
1186 * unmarked(
1187 * 0) result for corresponding bufPtrArray entry
1188 * ****
1189 *
1190 * eerrno_ty EDMRST_IsObjectMarked( serverHandle svrHdl,
1191 * const unsigned long numEntries,
1192 * restoreableObjectPtr *bufPtrArray,
1193 * unsigned long *numChecked,
1194 * boolean_ty *marked );
1195 *
1196 * ****
1197 * eerrno_ty EDMRST_GetMarkedTotalSize( serverHandle svrHdl,
1198 * u_hyper *totalSize );
1199 *
1200 * ****
1201 * Get Necessary Media:
1202 *
1203 * This function is provided to allow retrieval of the
1204 * media necessary to restore the currently marked objects.
1205 * The cookie must be initialize to INIT_COOKIE on the first call to
1206 * this
1207 * routine.
1208 * This routine will update the cookie to allow retrieval of more
1209 * objects if there is more than "maxEntries". The cookie will be
1210 * returned as DONE_COOKIE when there are no more to retrieve.
1211 * Parameters:
1212 * svrHdl (
1213 * I) - a pointer to this user's client handle for the
1214 * Restore Engine (server) connection.
1215 * maxEntries (I) - the maximum number of media objects to return
1216 * objects (O) - a pre-allocated array to return the objects in
1217 * numberEntries (
1218 * O) - the real number of media objects returned in the array
1219 * (I) - get all media, or necessary
1220 * (IO) - a place holder for the list position
1221 * all
1222 * cookie
1223 * ****
1224 * eerrno_ty EDMRST_GetNecessaryMedia( serverHandle svrHdl,
1225 * const short maxEntries,
1226 * mediaObjectPtr *objects,
1227 * short *numberEntries,
1228 * boolean_ty all,
1229 * long *cookie );
1230 *
1231 * ****
1232 * Media Object Access Routines:
1233 *
1234 * These routines retrieve information pertinent to a given Media
1235 * object.
1236 *
1237 * Parameters:
1238 *
1239 * Fri Oct 10 14:57:43 2008 restore_aplh.22 Page 22 of 36

```

1233	*	<i>svrHdl</i>	(I) - (restore aul.h 23	Page 23 of 36
1234	*	ignored	A pointer to this user's client handle for the		
1235	*	<i>thisObject</i>	(I) - The media object		
1236	*	*****	*****		
1237	*	const char	*EDMRST_GetMediaValid(serverHandle svrHdl,		
1238	1239	mediaObjectPtr	thisObject);		
1240	1241	long EDMRST_GetMediaSequenceNumber(serverHandle svrHdl,	mediaObjectPtr thisObject);		
1242	1243	const char	*EDMRST_GetMediaBarcodeString(serverHandle svrHdl,		
1245	1246	mediaObjectPtr	thisObject);		
1248	1249	const char	*EDMRST_GetMediaTypedescr(serverHandle svrHdl,		
1251	1252	mediaObjectPtr	thisObject);		
1254	1255	const char	*EDMRST_GetMediaToken(serverHandle svrHdl,		
1257	1258	mediaObjectPtr	thisObject);		
1260	1261	const char	*EDMRST_GetMediaComments(serverHandle svrHdl,		
1263	1264	mediaObjectPtr	thisObject);		
1266	1267	MediaStatus	EDMRST_GetMediaStatus(serverHandle svrHdl,		
1269	1270	mediaObjectPtr	thisObject);		
1271	1272	uchar_t EDMRST_GetMediaSide(serverHandle svrHdl,	mediaObjectPtr thisObject);		
1273	1274	*****	*****		
1275	1276	/* Duplicate Media Access Routines			
1277	1278	* Inputs: Svr Handle - see above			
1279	1280	dup_number: the number of the duplicate wanted			
1281	1282	thisObject: the media object to get the dups for...			
1283	1284	*****	*****		
1285	1286	short EDMRST_GetNumberOfDuplicats(serverHandle svrHdl,	mediaObjectPtr thisObject);		
1287	1288	const char *			
1289	1290	EDMRST_GetDuplicateValid(serverHandle svrHdl,	int dup_number,		
1291	1292	mediaObjectPtr	thisObject);		
1293	1294	long			
1295	1296	EDMRST_GetDuplicateSequenceNumber(serverHandle svrHdl,	int dup_number,		
1297	1298	mediaObjectPtr	thisObject);		
1299	1300	const char *			
1301	1302	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1303	1304	mediaObjectPtr	thisObject);		
1305	1306	const char *			
1307	1308	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1309	1310	mediaObjectPtr	thisObject);		
1311	1312	const char *			
1313	1314	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1315	1316	mediaObjectPtr	thisObject);		
1317	1318	const char *			
1319	1320	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1321	1322	mediaObjectPtr	thisObject);		
1323	1324	const char *			
1325	1326	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1327	1328	mediaObjectPtr	thisObject);		
1329	1330	const char *			
1331	1332	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1333	1334	mediaObjectPtr	thisObject);		
1335	1336	const char *			
1337	1338	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1339	1340	mediaObjectPtr	thisObject);		
1341	1342	const char *			
1343	1344	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1345	1346	mediaObjectPtr	thisObject);		
1347	1348	const char *			
1349	1350	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1351	1352	mediaObjectPtr	thisObject);		
1353	1354	const char *			
1355	1356	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1357	1358	mediaObjectPtr	thisObject);		
1359	1360	const char *			
1361	1362	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1363	1364	mediaObjectPtr	thisObject);		
1365	1366	const char *			
1367	1368	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1369	1370	mediaObjectPtr	thisObject);		
1371	1372	const char *			
1373	1374	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1375	1376	mediaObjectPtr	thisObject);		
1377	1378	const char *			
1379	1380	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1381	1382	mediaObjectPtr	thisObject);		
1383	1384	const char *			
1385	1386	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1387	1388	mediaObjectPtr	thisObject);		
1389	1390	const char *			
1391	1392	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1393	1394	mediaObjectPtr	thisObject);		
1395	1396	const char *			
1397	1398	EDMRST_GetDuplicateBarcodeString(serverHandle svrHdl,	int dup_number,		
1399</					

1298	EDMRST_GetDuplicateTypeDescrip(serverHandle svrhdl,
1299	int dup_number,
1300	mediaObjectPtr thisObject);
1303	const char *
1304	EDMRST_GetDuplicateTokenType(serverHandle svrhdl,
1305	int dup_number,
1306	mediaObjectPtr thisObject);
1309	MediaStatus
1310	EDMRST_GetDuplicateStatus(serverHandle svrhdl,
1311	int dup_number,
1312	mediaObjectPtr thisObject);
1313	const char *
1314	EDMRST_GetDuplicateLocation(serverHandle svrhdl,
1315	int dup_number,
1316	mediaObjectPtr thisObject);
1318	/******
1319	* Restoral Management Functions:
1320	*
1321	* These functions are provided to allow:
1322	* - creation of submit objects,
1323	* restored and the scripts to be run before and after
1324	* - starting the restoral of a submit object.
1325	* - polling of the status of an ongoing restore,
1326	* interrupt the restore,
1327	* and to receive information necessary to
1328	* query the user for input needed for the pre-restore or
1329	* - supply of user responses to pre- and post- restore script
1330	* queries
1331	* *
1332	* The following functions comprise restoral management:
1333	* *
1334	* EDMRST_Submit
1335	* EDMRST_GetSubmitResults
1336	* EDMRST_Start
1337	* EDMRST_GetRestoreFeedback
1338	* EDMRST_GetQuestion
1339	* EDMRST_SetUserAnswer
1340	* *****
1341	* Submit
1342	* *
1343	* This function starts the creation or update of a submit object from
1344	* currently marked restorable objects.
1345	* Its completion is tested for with
1346	* EDMRST_GetSubmitResults.
1347	* The returned submit object ID is passed to
1348	* EDMRST_Start to begin execution of the restore.
1349	* Parameters:
1350	* svrhdl (I) - A pointer to this user's client handle for
1351	* the Restore Engine (server) connection.
1352	* policy (I) - The overwrite policy to use
1353	* inplace

```
1354 * hostname (I) - host to restore to (only if inplace == False)
1355 * directory (I) - directory to restore to (
      * only if inplace == False)
1356 * transport (
      * I) - Indicator of transport the restoral is to be over (SCSI
      * or network)
1357 * submitObjID (
      * I) - ID of an existing submit object which is to be added to
1358
1359 * Return Codes:
1360 * E_SUCCESS - operation completed successfully
1361 * EP_RB_RECOVER_BAD_ARGS
1362 * EP_RB_RECOVER_NOMEM
1363 * EP_RB_RECOVER_RPC_FAIL
1364 * EP_RB_RECOVER_SERVER_FAIL
1365 * EP_RB_RECOVER_INVALIDOP
1366
      * -If another request still executing
      *****/
1367 eerrno_tly EDMRST_Submit( serverHandle
      const char svrHdl,
1368                      const char *hostname,
1369                      const OverwritePolicy policy,
1370                      const boolean_t inplace,
1371                      const char *directory,
1372                      const RestoreTransport transport,
1373                      submitObjID,
1374                      unsigned int *submitArgs);
1375
1376 /*****
1377 * GetSubmitResults
1378
1379 * This function tests for completion of an EDMRST_Submit call,
1380 * with the
      * option of cancelling the submit.
1381
1382 * Parameters:
1383 * svrHdl (I) - A pointer to this user's client handle for
1384 * the Restore Engine (server) connection.
1385 * interrupt (I) - Flag if the submit is to be canceled
1386 * submitObjID (
1387 * O) - ID of the submit object which describes the restore
1388 * objectsDone (
1389 * O) - number of objects -- total number in the submit object
1390 * if operation is complete,
      * or number processed so far if
      * submit operation is still executing (
      * INCOMPLETE status)
1391
1392 * Return Codes:
1393 * E_SUCCESS - operation completed successfully
1394 * EP_RB_RECOVER_RPC_INCOMPLETE
1395 * EP_RB_RECOVER_ABORT
1396 * EP_RB_RECOVER_FATALERR
1397
      * -If submit not done yet
      * -If submit successfully
      * -If submit interrupted
      * -If submit failed.
      * Internal restore error
1398 * EP_RB_RECOVER_BAD_ARGS
1399 * EP_RB_RECOVER_NOMEM
1400 * EP_RB_RECOVER_RPC_FAIL
1401 * EP_RB_RECOVER_SERVER_FAIL
1402 * EP_RB_RECOVER_INVALIDOP
1403
      * -If another request still
      * executing
      *****/
1403 eerrno_tly EDMRST_GetSubmitResults( serverHandle svrHdl,
1404                      const boolean_t interrupt,
1405                      unsigned int *submitObjID,
1406                      unsigned long *objectsDone );
1407
      restore_aplh 25 Page 25 of 36
```

```
1410 /*****
1411 * Start
1412
1413 * This function begins execution of the restoral of the objects in a
1414 * submit object. Its progress and requests for operator input are
1415 * received via EDMRST_GetRestoreFeedback.
1416
1417 * Parameters:
1418 * svrHdl (I) - A pointer to this user's client handle for
1419 * the Restore Engine (server) connection.
1420 * submitObjID (
1421 * I) - ID of the submit object that describes the restore
1422
1423 * Return Codes:
1424 * E_SUCCESS - operation completed successfully
1425 * EP_RB_RECOVER_BAD_ARGS
1426 * EP_RB_RECOVER_NOMEM
1427 * EP_RB_RECOVER_RPC_FAIL
1428 * EP_RB_RECOVER_SERVER_FAIL
1429 * EP_RB_RECOVER_INVALIDOP
1430
      * -If another request still
      * executing
      *****/
1430 eerrno_tly EDMRST_Start( serverHandle svrHdl,
1431                      unsigned int submitObjID );
1432
1433 /*****
1434 * GetRestoreFeedback
1435
1436 * This function is used to poll for the status of an ongoing restore,
1437 * and
      * includes the ability to interrupt the restore,
      * and to receive information
      * necessary to query the user for input needed for the pre-restore or
      * post-restore scripts.
1438
1439 * Parameters:
1440 * svrHdl (I) - A pointer to this user's client handle for
1441 * the Restore Engine (server) connection.
1442 * currentState (
1443 * O) - Pointer to storage to receive the state of the restore
1444 * feedbackPtr (
1445 * O) - Pointer to structure to receive restore feedback data
1446
1447 * Return Codes:
1448 * E_SUCCESS - If restore operation completed
1449 * EP_RB_RECOVER_RPC_INCOMPLETE
1450 * EP_RB_RECOVER_ABORT
1451 * EP_RB_RECOVER_FATALERR
1452 * EP_RB_RECOVER_EXECUTEFAILED
1453 * EP_RB_RECOVER_PREFAILED
1454 * EP_RB_RECOVER_POSTFAILED
1455 * EP_RB_RECOVER_CLIENT_FAIL
1456
      * -If restore not done yet
      * -If restore successfully
      * -If restore interrupted
      * -If restore execution failed
      * -If pre-restore
      * script execution failed
      * -If post-restore script
      * execution failed
      * -If problem with restore
      * target host
1457
1458 * EP_RB_RECOVER_BAD_ARGS
1459 * EP_RB_RECOVER_NOMEM
1460 * EP_RB_RECOVER_RPC_FAIL
1461 * EP_RB_RECOVER_SERVER_FAIL
1462 * EP_RB_RECOVER_INVALIDOP
1463
      * -If another request still
      *****/
      restore_aplh 26 Page 26 of 36
```

```

1463 *****
1465 eerrno_ty EDMRST_GetRestoreFeedback( serverHandle svrHdl,
1466 const boolean_ty quitRestore,
1467 RERunningState *currentState,
1468 feedbackObjectPtr feedbackPtr );
1470 /*****
1471 * GetQuestion
1472 * This function is used to fetch the data needed to query the user
1473 * during a
1474 * pre-restore or post-restore script execution.
1475 * Parameters:
1476 * svrHdl (I) - A pointer to this user's client handle for
1477 * the Restore Engine (server) connection.
1478 * queryPtr (O) - Pointer to the object containing the question data.
1480 * Return Codes:
1481 * E_SUCCESS - operation completed successfully
1482 * EP_RB_RECOVER_BAD_ARGS - If no question awaiting
1483 * EP_RB_RECOVER_RPC_FAIL
1484 * EP_RB_RECOVER_SERVER_FAIL
1485 * EP_RB_RECOVER_INVALIDOP
1486 *****
1487 eerrno_ty EDMRST_GetQuestion( serverHandle svrHdl,
1490 queryObjectPtr queryPtr );
1491 /*****
1493 * SetUserAnswer
1494 * This function is used to return user input requested via the
1495 * feedbackPtr
1496 * parameter output of the GetRestoreFeedback function call.
1497 * Parameters:
1498 * svrHdl (I) - A pointer to this user's client handle for
1499 * the Restore Engine (server) connection.
1500 * QueryPtr (I) - Pointer to structure containing the query and response
1501 * data.
1502 * answer (I) - pointer to text string response to question.
1503 * more (I) - indicator that there will be more answers to this question
1504 * Return Codes:
1505 * E_SUCCESS - operation completed successfully
1506 * EP_RB_RECOVER_BAD_ARGS
1507 * EP_RB_RECOVER_NOMEM
1508 * EP_RB_RECOVER_RPC_FAIL
1509 * EP_RB_RECOVER_SERVER_FAIL
1510 * EP_RB_RECOVER_INVALIDOP -If no question awaiting
1511 *****
1512 eerrno_ty EDMRST_SetUserAnswer( serverHandle svrHdl,
1513 queryObjectPtr queryPtr,
1514 const char *answer,
1515 boolean_ty more );
1516
1517 restore_apih 27
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000

```

```

1524 /*****
1525 * Query Object Access Routines:
1526 * These routines retrieve portions of a specified Query object.
1527 * object is returned by EDMRST_GetQuestion,
1528 * and is used to get a response
1529 * From the Restore API 'operator' after a restore has been started.
1530 * Parameters common to all these functions:
1531 * svrHdl (I) - A pointer to this user's client handle for the
1532 * Restore Engine (
1533 * server) connection. Must be valid.
1534 * thisObject (I) - The query object
1535 * Parameters specific to only one function:
1536 * maxlen (O) - minimum response length
1537 * maxlen (O) - maximum response length
1538 * isset
1539 * cookie (O) - boolean_ty indicating this is the default response
1540 * cookie (IO) - placeholder in list of possible choices
1541 * Returns one of the following:
1542 * int question type or number of possible responses;
1543 * const char * ptr to a text string in the query object,
1544 * NULL on errs
1545 * eerrno_ty E_SUCCESS, or EP_RB_RECOVER_XXX on errors
1546 *****
1547 int EDMRST_GetQuestionType( serverHandle svrHdl,
1548 queryObjectPtr thisObject );
1549
1550 eerrno_ty EDMRST_GetQuestionAnswersSize( serverHandle svrHdl,
1551 queryObjectPtr thisObject,
1552 int *minlen,
1553 int *maxlen );
1554
1555 int EDMRST_GetQuestionNumChoices( serverHandle svrHdl,
1556 queryObjectPtr thisObject );
1557
1558 const char * EDMRST_GetQuestionNextChoice(
1559 serverHandle svrHdl,
1560 queryObjectPtr thisObject,
1561 int *isset,
1562 long *cookie );
1563
1564 const char * EDMRST_GetQuestionInvalidChars(
1565 serverHandle svrHdl,
1566 queryObjectPtr thisObject );
1567
1568 const char * EDMRST_GetQuestionHeaderText(
1569 serverHandle svrHdl,
1570 queryObjectPtr thisObject );
1571
1572 const char * EDMRST_GetQuestionText( serverHandle svrHdl,
1573 queryObjectPtr thisObject );
1574
1575 restore_apih 28
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900

```

```

1575 /***** */
1576 /** EDMRST_GetHostPlatformType */
1577 /*/
1578 /** This function returns the platform type of the
1579 /** specified host. */
1580 /** */
1581 /** Parameters: */
1582 /** svrHdl - (I) A pointer to this user's client
1583 /** handle for the Restore Engine
1584 /** */
1585 /** pHostName - (I) ptr to hostname string;
1586 /** pType - (O) ptr to host platform type, which
1587 /** can be one of the following valid
1588 /** types: */
1589 /** BUCFG_PLATFORM_UNIX */
1590 /** BUCFG_PLATFORM_OS2 */
1591 /** BUCFG_PLATFORM_WNT */
1592 /** BUCFG_PLATFORM_NETWORKWARE */
1593 /** */
1594 /** */
1595 /** Return Codes: */
1596 /** E_SUCCESS: */
1597 /** - operation successful, *pType will
1598 /** contain one of the valid types
1599 /** described above. */
1600 /** */
1601 /** EP_RB_RECOVER_BAD_ARGS */
1602 /** EP_RB_RECOVER_RPC_FAIL */
1603 /** EP_RB_RECOVER_SERVER_FAIL */
1604 /** EP_RB_RECOVER_INVALIDOP - If another request active
1605 /** */
1606 /** */
1607 /***** */
1608 /** eerrno_ty EDMRST_GetHostPlatformType( serverHandle svrHdl,
1609 /** const char *pHostName,
1610 /** BUCfgPlatformType *pType ); */
1611 /***** */
1612 /***** */
1613 /***** */
1614 /***** */
1615 /** EDMRST_GetBackupTimesSupport */
1616 /** */
1617 /** Function Description: */
1618 /** Determine if the specified top level object can be restored
1619 /** from multiple backup times. */
1620 /** */
1621 /** Parameters: */
1622 /** svrHdl (I) - A pointer to this user's client handle for
1623 /** the Restore Engine (server) connection.
1624 /** thisObject (I) - The restoral object
1625 /** isSupported (I) - TRUE/FALSE that restores from multiple backup planes
1626 /** O) - TRUE/FALSE that restores from multiple backup planes
1627 /** is/is not supported */
1628 /** */
1629 /** Return Codes: */
1630 /** E_SUCCESS - operation completed successfully
1631 /** EP_RB_RECOVER_BAD_ARGS */
1632 /** EP_RB_RECOVER_RPC_FAIL */
1633 /** EP_RB_RECOVER_SERVER_FAIL */
1634 /** EP_RB_RECOVER_INVALIDOP -If another request active
1635 /** */
1636 /***** */
1637 /***** */
1638 /***** */
1639 /***** */
1640 /***** */
1641 /***** */
1642 /***** */
1643 /***** */
1644 /***** */
1645 /***** */
1646 /***** */
1647 /***** */
1648 /***** */
1649 /***** */
1650 /***** */
1651 /***** */
1652 /***** */
1653 /***** */
1654 /***** */
1655 /***** */
1656 /***** */
1657 /***** */
1658 /***** */
1659 /***** */
1660 /***** */
1661 /***** */
1662 /***** */
1663 /***** */
1664 /***** */
1665 /***** */
1666 /***** */
1667 /***** */
1668 /***** */
1669 /***** */
1670 /***** */
1671 /***** */
1672 /***** */
1673 /***** */
1674 /***** */
1675 /***** */
1676 /***** */
1677 /***** */
1678 /***** */
1679 /***** */
1680 /***** */
1681 /***** */
1682 /***** */
1683 /***** */
1684 /***** */
1685 /***** */
1686 /***** */
1687 /***** */
1688 /***** */
1689 /***** */
1690 /***** */
1691 /***** */
1692 /***** */
1693 /***** */
1694 /***** */
1695 /***** */
1696 /***** */
1697 /***** */
1698 /***** */
1699 /***** */
1700 /***** */
1701 /***** */
1702 /***** */
1703 /***** */
1704 /***** */
1705 /***** */
1706 /***** */
1707 /***** */
1708 /***** */
1709 /***** */
1710 /***** */
1711 /***** */
1712 /***** */
1713 /***** */
1714 /***** */
1715 /***** */
1716 /***** */
1717 /***** */
1718 /***** */
1719 /***** */
1720 /***** */
1721 /***** */
1722 /***** */
1723 /***** */
1724 /***** */
1725 /***** */
1726 /***** */
1727 /***** */
1728 /***** */
1729 /***** */
1730 /***** */
1731 /***** */
1732 /***** */
1733 /***** */
1734 /***** */
1735 /***** */
1736 /***** */
1737 /***** */
1738 /***** */
1739 /***** */
1740 /***** */
1741 /***** */
1742 /***** */
1743 /***** */
1744 /***** */
1745 /***** */
1746 /***** */
1747 /***** */
1748 /***** */
1749 /***** */
1750 /***** */
1751 /***** */
1752 /***** */
1753 /***** */
1754 /***** */
1755 /***** */
1756 /***** */
1757 /***** */
1758 /***** */
1759 /***** */
1760 /***** */
1761 /***** */
1762 /***** */
1763 /***** */
1764 /***** */
1765 /***** */
1766 /***** */
1767 /***** */
1768 /***** */
1769 /***** */
1770 /***** */
1771 /***** */
1772 /***** */
1773 /***** */
1774 /***** */
1775 /***** */
1776 /***** */
1777 /***** */
1778 /***** */
1779 /***** */
1780 /***** */
1781 /***** */
1782 /***** */
1783 /***** */
1784 /***** */
1785 /***** */
1786 /***** */
1787 /***** */
1788 /***** */
1789 /***** */
1790 /***** */
1791 /***** */
1792 /***** */
1793 /***** */
1794 /***** */
1795 /***** */
1796 /***** */
1797 /***** */
1798 /***** */
1799 /***** */
1800 /***** */
1801 /***** */
1802 /***** */
1803 /***** */
1804 /***** */
1805 /***** */
1806 /***** */
1807 /***** */
1808 /***** */
1809 /***** */
1810 /***** */
1811 /***** */
1812 /***** */
1813 /***** */
1814 /***** */
1815 /***** */
1816 /***** */
1817 /***** */
1818 /***** */
1819 /***** */
1820 /***** */
1821 /***** */
1822 /***** */
1823 /***** */
1824 /***** */
1825 /***** */
1826 /***** */
1827 /***** */
1828 /***** */
1829 /***** */
1830 /***** */
1831 /***** */
1832 /***** */
1833 /***** */
1834 /***** */
1835 /***** */
1836 /***** */
1837 /***** */
1838 /***** */
1839 /***** */
1840 /***** */
1841 /***** */
1842 /***** */
1843 /***** */
1844 /***** */
1845 /***** */
1846 /***** */
1847 /***** */
1848 /***** */
1849 /***** */
1850 /***** */
1851 /***** */
1852 /***** */
1853 /***** */
1854 /***** */
1855 /***** */
1856 /***** */
1857 /***** */
1858 /***** */
1859 /***** */
1860 /***** */
1861 /***** */
1862 /***** */
1863 /***** */
1864 /***** */
1865 /***** */
1866 /***** */
1867 /***** */
1868 /***** */
1869 /***** */
1870 /***** */
1871 /***** */
1872 /***** */
1873 /***** */
1874 /***** */
1875 /***** */
1876 /***** */
1877 /***** */
1878 /***** */
1879 /***** */
1880 /***** */
1881 /***** */
1882 /***** */
1883 /***** */
1884 /***** */
1885 /***** */
1886 /***** */
1887 /***** */
1888 /***** */
1889 /***** */
1890 /***** */
1891 /***** */
1892 /***** */
1893 /***** */
1894 /***** */
1895 /***** */
1896 /***** */
1897 /***** */
1898 /***** */
1899 /***** */
1900 /***** */
1901 /***** */
1902 /***** */
1903 /***** */
1904 /***** */
1905 /***** */
1906 /***** */
1907 /***** */
1908 /***** */
1909 /***** */
1910 /***** */
1911 /***** */
1912 /***** */
1913 /***** */
1914 /***** */
1915 /***** */
1916 /***** */
1917 /***** */
1918 /***** */
1919 /***** */
1920 /***** */
1921 /***** */
1922 /***** */
1923 /***** */
1924 /***** */
1925 /***** */
1926 /***** */
1927 /***** */
1928 /***** */
1929 /***** */
1930 /***** */
1931 /***** */
1932 /***** */
1933 /***** */
1934 /***** */
1935 /***** */
1936 /***** */
1937 /***** */
1938 /***** */
1939 /***** */
1940 /***** */
1941 /***** */
1942 /***** */
1943 /***** */
1944 /***** */
1945 /***** */
1946 /***** */
1947 /***** */
1948 /***** */
1949 /***** */
1950 /***** */
1951 /***** */
1952 /***** */
1953 /***** */
1954 /***** */
1955 /***** */
1956 /***** */
1957 /***** */
1958 /***** */
1959 /***** */
1960 /***** */
1961 /***** */
1962 /***** */
1963 /***** */
1964 /***** */
1965 /***** */
1966 /***** */
1967 /***** */
1968 /***** */
1969 /***** */
1970 /***** */
1971 /***** */
1972 /***** */
1973 /***** */
1974 /***** */
1975 /***** */
1976 /***** */
1977 /***** */
1978 /***** */
1979 /***** */
1980 /***** */
1981 /***** */
1982 /***** */
1983 /***** */
1984 /***** */
1985 /***** */
1986 /***** */
1987 /***** */
1988 /***** */
1989 /***** */
1990 /***** */
1991 /***** */
1992 /***** */
1993 /***** */
1994 /***** */
1995 /***** */
1996 /***** */
1997 /***** */
1998 /***** */
1999 /***** */
2000 /***** */
2001 /***** */
2002 /***** */
2003 /***** */
2004 /***** */
2005 /***** */
2006 /***** */
2007 /***** */
2008 /***** */
2009 /***** */
2010 /***** */
2011 /***** */
2012 /***** */
2013 /***** */
2014 /***** */
2015 /***** */
2016 /***** */
2017 /***** */
2018 /***** */
2019 /***** */
2020 /***** */
2021 /***** */
2022 /***** */
2023 /***** */
2024 /***** */
2025 /***** */
2026 /***** */
2027 /***** */
2028 /***** */
2029 /***** */
2030 /***** */
2031 /***** */
2032 /***** */
2033 /***** */
2034 /***** */
2035 /***** */
2036 /***** */
2037 /***** */
2038 /***** */
2039 /***** */
2040 /***** */
2041 /***** */
2042 /***** */
2043 /***** */
2044 /***** */
2045 /***** */
2046 /***** */
2047 /***** */
2048 /***** */
2049 /***** */
2
```

[illegible]

```
1697 *
1698 * Return Codes:
1699 * E_SUCCESS - operation completed successfully
1700 * EP_RB_RECOVER_BAD_ARGS
1701 * EP_RB_RECOVER_RPC_FAIL
1702 * EP_RB_RECOVER_SERVER_FAIL
1703 * EP_RB_RECOVER_INVALIDOP
1704 * EP_RB_RECOVER_NO_SAVESSET
1705 * EP_RB_RECOVER_NO_CATALOG - If another request active
1706 * - when errors occur accessing
1707 * savesets
1708 * catalogs
1709 *
1710 * EDMRST_IsThereNextBackupForTime( serverHandle, svrHdl,
1711 * u_long const time_t, thisTime, flags,
1712 * boolean_ty *isThere );
1713
1714 /*****
1715 * EDMRST_IsTherePrevBackupForTime
1716 *
1717 * Function Description:
1718 * Determine if a backup exists prior to the specified time
1719 *
1720 * Parameters:
1721 * svrHdl (I) - A pointer to this user's client handle for
1722 * the Restore Engine (server) connection.
1723 * thisTime(I) - Time for the query
1724 * flags (I) - Backup constraint flags: e.g., full-only/partial-ok
1725 * isThere (O) - TRUE/FALSE that requested backup does exist
1726 *
1727 * Return Codes:
1728 * E_SUCCESS - operation completed successfully
1729 * EP_RB_RECOVER_BAD_ARGS
1730 * EP_RB_RECOVER_RPC_FAIL
1731 * EP_RB_RECOVER_SERVER_FAIL
1732 * EP_RB_RECOVER_INVALIDOP - If another request active
1733 * EP_RB_RECOVER_NO_SAVESSET - when errors occur accessing
1734 * EP_RB_RECOVER_NO_CATALOG - when errors occur accessing
1735 * savesets
1736 * catalogs
1737 *
1738 *
1739 * EDMRST_IsTherePrevBackupForTime( serverHandle, svrHdl,
1740 * const time_t, thisTime, flags,
1741 * u_long boolean_ty *isThere );
1742 *
1743 /*****
1744 * EDMRST_IsThereNextBackupForTime
1745 *
1746 * Function Description:
1747 * Determine if a backup exists after to the specified time
1748 *
1749 * Parameters:
1750 * context (I) - Pointer to the restore context
1751 * thisTime(I) - Time for the query
1752 * flags (I) - Backup constraint flags: e.g., full-only/partial-ok
1753 * isThere (O) - TRUE/FALSE that requested backup does exist
1754 *
```

```
1755 *
1756 * Return Codes:
1757 * E_SUCCESS - operation completed successfully
1758 * EP_RB_RECOVER_BAD_ARGS
1759 * EP_RB_RECOVER_RPC_FAIL
1760 * EP_RB_RECOVER_SERVER_FAIL
1761 * EP_RB_RECOVER_INVALIDOP
1762 * EP_RB_RECOVER_NO_SAVESSET
1763 * EP_RB_RECOVER_NO_CATALOG - If another request active
1764 * - when errors occur accessing
1765 * savesets
1766 * catalogs
1767 *
1768 * EDMRST_IsThereNextBackupForTime( serverHandle, svrHdl,
1769 * const time_t, thisTime, flags,
1770 * boolean_ty *isThere );
1771
1772 /*****
1773 * EDMRST_IsTherePrevBackupForTime
1774 *
1775 * Function Description:
1776 * Determine if a backup exists prior to the specified time
1777 *
1778 * Parameters:
1779 * svrHdl (I) - A pointer to this user's client handle for the
1780 * Restore Engine (server) connection. Must be valid.
1781 * thisTime(I) - Time for the query
1782 * flags (I) - Backup constraint flags: e.g., full-only/partial-ok
1783 * isThere (O) - TRUE/FALSE that requested backup does exist
1784 *
1785 * Return Codes:
1786 * E_SUCCESS - operation completed successfully
1787 * EP_RB_RECOVER_BAD_ARGS
1788 * EP_RB_RECOVER_RPC_FAIL
1789 * EP_RB_RECOVER_SERVER_FAIL
1790 * EP_RB_RECOVER_INVALIDOP - If another request active
1791 * EP_RB_RECOVER_NO_SAVESSET - when errors occur accessing
1792 * EP_RB_RECOVER_NO_CATALOG - when errors occur accessing
1793 * savesets
1794 * catalogs
1795 *
1796 *
1797 * EDMRST_IsTherePrevBackupForTime( serverHandle, svrHdl,
1798 * const time_t, thisTime, flags,
1799 * u_long boolean_ty *isThere );
1800 *
1801 /*****
1802 * EDMRST_IsThereNextBackupForTime
1803 *
1804 * Function Description:
1805 * Determine if a backup exists after to the specified time
1806 *
1807 * Parameters:
1808 * context (I) - Pointer to the restore context
1809 * thisTime(I) - Time for the query
1810 * flags (I) - Backup constraint flags: e.g., full-only/partial-ok
1811 * isThere (O) - TRUE/FALSE that requested backup does exist
1812 *
```

```

1802 u_long EDMRST_GetObjectTotalKBBytesSoFar(
1803     u_long EDMRST_GetObjectTotalFiles(
1804         u_long EDMRST_GetObjectTotalBadFiles(
1805             u_long EDMRST_GetObjectCurKBBytesSoFar(
1806                 u_long EDMRST_GetObjectCurTimeSlice(
1807                     u_long EDMRST_GetObjectCurFiles(
1808                         u_long EDMRST_GetObjectTotalFilesExpected(
1809                             u_long EDMRST_GetObjectTotalKBExpected(
1810                                 u_long EDMRST_GetObjectItemStatus(
1811                                     int EDMRST_GetObjectOperationType(
1812                                         int EDMRST_GetObjectCompleted(
1813                                             int EDMRST_IsLastObject(serverHandle svrHdl, WIPProgressPtr thisObject);
1814 WIPProgressPtr EDMRST_GetNextObject(
1815     WIPProgressPtr EDMRST_GetFirstObject(
1816         serverHandle svrHdl, feedbackObjectPtr thisObject);
1817 EDMProgressPtr
1818 EDMRST_GetFirstEDMObject(
1819     EDMProgressPtr
1820 EDMRST_GetNextEDMObject(
1821     serverHandle svrHdl, EDMProgressPtr thisObject);
1822 u_long EDMRST_GetObjectEDMTimeStarted(
1823     serverHandle svrHdl, EDMProgressPtr thisObject);
1824 u_long EDMRST_GetObjectEDMCurTime(
1825     serverHandle svrHdl, EDMProgressPtr thisObject);
1826 EDMRST_GetObjectEDMTotalKBBytesSoFar(
1827     serverHandle svrHdl, EDMProgressPtr thisObject);
1828 u_long EDMRST_GetObjectEDMTotalFiles(
1829     serverHandle svrHdl, EDMProgressPtr thisObject);
1830 EDMRST_GetObjectEDMTotalBadFiles(
1831     serverHandle svrHdl, EDMProgressPtr thisObject);
1832 u_long EDMRST_GetObjectEDMCurTimeSlice(
1833     serverHandle svrHdl, EDMProgressPtr thisObject);
1834 EDMRST_GetObjectEDMCurKBBytesSoFar(
1835     serverHandle svrHdl, EDMProgressPtr thisObject);
1836 EDMRST_GetObjectEDMCurFiles(
1837     serverHandle svrHdl, EDMProgressPtr thisObject);
1838 EDMRST_GetObjectEDMActive(
1839     serverHandle svrHdl, EDMProgressPtr thisObject);
1840 EDMRST_GetObjectEDMTotal(
1841     serverHandle svrHdl, EDMProgressPtr thisObject);
1842 EDMRST_GetObjectEDMFailed(
1843     serverHandle svrHdl, EDMProgressPtr thisObject);

```

```

1843 u_long EDMRST_GetObjectEDMSuccessful(
1844     serverHandle svrHdl, EDMProgressPtr thisObject);
1845 u_long EDMRST_GetObjectEDMTotalFilesExpected(
1846     serverHandle svrHdl, EDMProgressPtr thisObject);
1847 u_long EDMRST_GetObjectEDMTotalKBExpected(
1848     serverHandle svrHdl, EDMProgressPtr thisObject);
1849 int EDMRST_GetObjectEDMOperationType(
1850     serverHandle svrHdl, EDMProgressPtr thisObject);
1851 int EDMRST_GetObjectEDMCompleted(
1852     serverHandle svrHdl, EDMProgressPtr thisObject);
1853 u_long EDMRST_GetObjectEDMStatus(
1854     serverHandle svrHdl, EDMProgressPtr thisObject);
1855 const char * EDMRST_GetObjectEDMHostName(
1856     serverHandle svrHdl, EDMProgressPtr thisObject);
1857 /** Notify Access Routines */
1858 void* EDMRST_GetFirstNotifyObject(
1859     serverHandle svrHdl, feedbackObjectPtr thisObject);
1860 void* EDMRST_GetNextNotifyObject(serverHandle svrHdl, void* thisObject);
1861 const char * EDMRST_GetNotifyMessageText(serverHandle svrHdl, void* thisObject);
1862 int EDMRST_GetNotifyMessageType(serverHandle svrHdl, void* thisObject);
1863 int EDMRST_GetNotifySourceModule(serverHandle svrHdl, void* thisObject);
1864 int EDMRST_GetNotifyLevel(serverHandle svrHdl, void* thisObject);
1865 int EDMRST_GetNotifyMessageLength(serverHandle svrHdl, void* thisObject);
1866 int EDMRST_IsLastNotifyObject(serverHandle svrHdl, void* thisObject);
1867
1868 /*****
1869 * SetRecxDirectives:
1870 *
1871 * This routine returns sends the filename and path plus hostname
1872 * of the recx directives file, which was created by the command
1873 * eb_dc_restore, to the server which then processes the recx
1874 * directives
1875 *
1876 * Parameters:
1877 *   svrHdl (I) - A pointer to this user's client handle for the
1878 *   Restore Engine (server) connection.
1879 *   template (O) - The name of the local recx file
1880 *   alternate (O) - the name of this host so the file can be transfered
1881 *
1882 *****/
1883 eerrno_t EDMRST_SetRecxDirectives( serverHandle svrHdl,
1884     char *filename,
1885     char *hostname );
1886
1887 /*****
1888 * EDMRST_get_catalag_info:
1889 *
1890 * This routine returns sends the fills the level string with the
1891 *****/

```

```
1900 * level for backup being restored
1901 *
1902 * Parameters:
1903 *   svrHdl          (I) - A pointer to this user's client handle for the
1904 *                   Restore Engine (server) connection.
1905 *   backup_time     (I) - Time of the backup that is being looked at
1906 *   *level          (O) - The level of the backup for specified time
1907 *                   taken from catalog structure. If not enough
1908 *                   memory has been allocated value will be "\0"
1909 *   *numrec         (O) - The number of records for the specified backup
1910 *                   taken from catalog structure. If not enough
1911 *                   memory has been allocated value will be "\0"
1912 *   *catType        (O) - The type of catalog for the specified backup
1913 *                   taken from catalog structure. If not enough
1914 *                   memory has been allocated value will be "\0"
1915 * Return Codes:
1916 *   EP_RB_RECOVER_BAD_ARGS - arguments passed in are null
1917 *   E_SUCCESS              - the fields have been filled in
1918 *                           and RPC succeeded
1919 *
1920 * *****/
1921 eerrno_ty      EDMRST_getCatalogInfo( serverHandle  svrHdl,
1922                                     time_t         backup_time,
1923                                     char            *level,
1924                                     char            *numRec,
1925                                     char            *catType);
1926
1927 #endif /* H_RESTOREAPI */
1928
```

D2

```

1  /*****
2  *      restoreRPC.h
3  *
4  *  Definitions of data sent between client and server for Restore API.
5  *
6  *  NOTE:
7  *  This file is intended to be shared by both RPC independent code on
8  *  the
9  *  client and server,
10 *  and by RPC implementation specific code. That is, it
11 *  is intended to be included in RPC definition files --
12 *  and in normal 'C' code.
13 *
14 *  This is so that the client and server code at all but the RPC call
15 *  service levels are RPC mechanism independent. HOWEVER,
16 *  definitions are very similar to OMC RPC data types (
17 *  e.g. STRING and OPAQUE),
18 *  and for another RPC implementation,
19 *  hopefully only a redefinition of those
20 *  two macros would be needed.
21 *  *****/
22 #ifndef H_RESTORE_RPC_DATA
23 #define H_RESTORE_RPC_DATA
24
25 /* Don't redefine this stuff if restore_engine.
26 * h has already been included */
27 #ifndef _RESTORE_ENGINE_H_RPCGEN
28
29 /* need to use string <> for .x file, char * for .h */
30 /* This doesn't work for some reason: #if RPC_HDR */
31 #ifdef IN_DOTX
32 #define STRING(x) string x<>
33 #define OPAQUE(x) opaque x<>
34 #else
35 #define STRING(x) char *x
36 #define OPAQUE(x) struct { u_int length; char *data; } x
37 #endif
38 #define RAW_NETWORK 0
39 #define PLUGIN 1
40
41 /*
42 * Bit field values for the Top Level Object "flags" field
43 */
44 typedef u_int RSTRPC_bool;
45 typedef unsigned long RSTRPC_backup_flags_ty;
46 typedef long RSTRPC_time_ty;
47
48 typedef int RSTRPC_enum_ty;
49
50 struct RSTRPC_u_hypr {
51     unsigned long high;
52     unsigned long low;
53 };
54
55 /* generic linked list structure definition: */
56 /* Linked lists used by the Restore Service Library must follow this
57    format --
58    * that is,
59
60 restoreRPC.h 1
61
62 Page 1 of 4

```

```

58 must begin with the link to the next list entry. The struct_ptr
59 * entry does NOT have to be a char *, but can be anything,
60 even a whole struct.
61
62 typedef char * struct_ptr;
63
64 struct RSTRPC_list_ty {
65     struct RSTRPC_list_ty *next;
66     struct_ptr objptr;
67 };
68
69 struct RSTRPC_time_list {
70     struct RSTRPC_time_list *next;
71     struct RSTRPC_time_list time;
72 };
73
74 struct RSTRPC_name_list {
75     struct RSTRPC_name_list *next;
76     STRING (name);
77 };
78
79 /* Structures for use in passing file name of the direct connect
80 * directives
81 */
82 struct RSTRPC_rex_file_info {
83     STRING (filename);
84     STRING (hostname);
85 };
86
87
88 /*
89 * Definition of restorableObject data components for shared use
90 * between
91 * generic (
92 *   ie not app-specific) client and server portions of Restore API.
93 *
94 * The restorableObject is presented to the users
95 * of the Restore API as an opaque object,
96 * meaning that its content and
97 * format are not known and not relevant to anyone except the Restore
98 * API internals. However, for Restore API internals,
99 * it must be shared by
100 * programs on the client and server side of the Restore API.
101 * The following
102 * structure is designed to be shared by both sides of the API,
103 * and to be
104 * backup-application independent.
105 * That is why the application-specific data
106 * within this structure is treated as opaque (
107 * long length and void *).
108 */
109
110 enum RSTRPC_BackupStatus {
111     RSTRPC_Backup_Good,
112     RSTRPC_Backup_Bad,
113     RSTRPC_Backup_Expired,
114     RSTRPC_Backup_Child_Without_Data
115 };
116
117 enum RSTRPC_ObjectLevel {
118     RSTRPC_tlo_type = 1,
119     RSTRPC_container_type,
120 };
121
122
123 Fri Oct 10 15:16:56 2008
124
125 restoreRPC.h 2
126
127 Page 2 of 4

```

```

114 1      RSTRPC_leaf_type
115 1      );
118      /* common part of all restorable objects: */
120 1      struct RSTRPC_restorable_obj_root {
121 1          enum RSTRPC_ObjectLevel      objLevel;
122 1          int      backupApp;
123 1          STRING   (objName);
124 1          STRING   (objTypeStr);
125 1      };
127      /* top level object data: in linked list,
      or in restorable object container */
129 1      struct RSTRPC_top_level_obj {
130 1          struct RSTRPC_restorable_obj_root root;
131 1          OPAQUE   (appData);
132 1          STRING   (hostname);
133 1          STRING   (fileSpec);
134 1          STRING   (templateName);
135 1          STRING   (wABIC);
136 1          char      wireType;
137 1          unsigned int flags;
138 1          RSTRPC_bool ssthread;
139 1      };
141 1      struct RSTRPC_tlo_list {
142 1          struct RSTRPC_tlo_list      *next;
143 1          struct RSTRPC_top_level_obj *tlo;
144 1      };
146 1      struct RSTRPC_user_restorable_object
147 1      {
148 1          struct RSTRPC_restorable_obj_root root;
149 1          OPAQUE   (appData);
150 1          u_long   objMode;
151 1          STRING   (objOwnerName);
152 1          STRING   (objGroupName);
153 1          RSTRPC_time_ty      objModTime;
154 1          /* Time obj was last modified */
155 1          struct RSTRPC_u_hyper objSize;
156 1          enum RSTRPC_BackupStatus objBackupStatus;
157 1          STRING   (objBaseName);
158 1          /* Base name of the object */
159 1      };
160 1      struct RSTRPC_uro_list {
161 1          struct RSTRPC_uro_list      *next;
162 1          struct RSTRPC_user_restorable_object *uro;
164 1      };
165 1      struct RSTRPC_found_obj_list {
166 1          struct RSTRPC_found_obj_list *next;
167 1          RSTRPC_time_ty      time;
168 1          struct RSTRPC_user_restorable_object *foundObj;
169 1      };
173 1      /*
174 1      * mediaobject definition
175 1      *
176 1      * The media object is an internal object of the Restore API.

```

```

177      * to the users of the Restore API as an opaque object.
178      * and format are not known except the Restore API internals.
179      * inquire about the media that must be read to restore a set of
180      * marked files.
181      *
182      struct RSTRPC_media_list {
183      struct RSTRPC_media_list      *next;
184      struct RSTRPC_media_object      *media_obj;
185      };
186      struct RSTRPC_media_object
187      {
188      STRING   (trail);
189      STRING   (mtype);
190      STRING   (mtype_token);
191      STRING   (barcode_label);
192      STRING   (physical_loc);
193      STRING   (comments);
194      STRING   (valid_ascii);
195      u_int     seqno;
196      u_char    side;
197      RSTRPC_time_ty lmtime;
198      RSTRPC_bool online;
199      RSTRPC_bool offline;
200      RSTRPC_bool is_orig;
201      RSTRPC_bool run_media_dup;
202      struct RSTRPC_media_list *dups;
203      short num_dups;
204      STRING   (luname);
205      };
208      #endif
210      #endif

```

D3

```

%/*
*** Copyright 1997,1998 EMC Corporation
%*/

/*
** Leading % causes rpcgen to pass a line directly thought to the output,
** ie restore_engine.h in this case. This allows the .h to make a little
** more sense and be properly documented.
*/

```

```

%/*
%* restore_engine.x : EDM Restore Engine C/S communication module
%*
%* Mission Statement: This is an RPCGEN file which defines the RPC interface
%* between the Restore Engine server (which resides on
%* the EDM server) and the backup client callers of its
%* functions. This defines the RPC level calls that a
%* "caller" can make and the "service" will respond to.
%*
%* Primary Data Acted On: This defines the data that will flow over the wire.
%* The RPC mechanism will take care of data
%* marshalling
%*
%*
%* Compile-Time Options:
%* This acutally gets run through RPCGEN not compiled. It
%* must be run through with the -h flag to create a
%* header, the -m flag to create the service side
%* routines, the -l flag to create the client side
%* routines, and the -c flag to create the common data
%* marshalling routines.
%*
%* Basic idea here:
%* Define the RPC level interfaces to the Restore Engine
%* and all data types that will be passed via RPC.
%*/

```

```

/* for sharing of STRING(x) and OPAQUE(x) */
#define IN_DOTX
#include <restore/restoreRPC.h>

#include <restore/dispatch_daemon.h>

/*****
Constant Definitions
*****/

/*****
Enum Definitions
*****/

/*****
Typedef Definitions
*****/

typedef int RE_errno_ty;

/*****
Data Structure Definitions
*****/

/* Structure to start every RPC request and response - for debug purposes */
struct RE_rpc_objID
{
    unsigned long rpc_type;
    restore_engine.x 1

```

```

/* RPC Object ID (ie, rpc #) */
RSTRPC_time_ty time; /* creation time */
long len; /* Length of structure, version num? */

```

```

};

struct RE_null_args {
    RE_rpc_objID RfcobjID;
};

```

```

struct RE_status_result {
    RE_rpc_objID RfcobjID;
    RE_errno_ty status;
};

```

```

struct RE_boolean_result {
    RE_rpc_objID RfcobjID;
    RE_errno_ty status;
    RSTRPC_bool boolResult;
};

```

```

union RE_restorable_obj switch (RSTRPC_ObjectLevel objLevel)
{
    case RSTRPC_tlo_type:
        RSTRPC_top_level_obj
        default: /* anything else means NOT tlo -- i.e. container or leaf */
            RSTRPC_user_restorable_object *uroinfo;
};

```

```

const MAX_CHOICE_TEXT=80;

struct Choices {
    bool isset;
    string ctext<>;
    Choices *nextchoice;
};

```

```

/* Question types */
const QTYPE_BOOL = 1;
const QTYPE_RAD = 2;
const QTYPE_MULTI = 4;
const QTYPE_STR = 8;
const QTYPE_YESNO = 16;
const QTYPE_INT = 32;

```

```

struct Question {
    int qnum;
    int qtype;
    int maxlen;
    int minlen;
    int numchoices;
    string invalidchars<>;
    string headertxt<>;
    string ctext<>;
    Choices *choices;
};

```

```

struct Answer {
    int qnum;
    string ctext<>;
    Answer *nextanswer;
};

```

```

struct Answerlist {
    int numanswers;
    Answer *firstanswer;
};

```

```

};

/* structures for input and output of re_initialize rpc call: */
struct RE_initialize_args {
    RE_rpc_objID RPCobjID;
    string username<>;
};

/* structures for input and output of get_source_hosts and
 * get_destination_hosts rpc calls:
 */
struct RE_get_hosts_args {
    RE_rpc_objID RPCobjID;
    string hostname<; /* only for get_source_hosts */
    short maxEntries;
    long cookie;

};

struct RE_get_hosts_result {
    RE_rpc_objID RPCobjID;
    RE_erno_ty status; /* redundant but useful ? */
    short numEntries;
    long cookie;
    RSTRPC_name_list *hosts; /* link to first hostname */
};

/* structure for single character string argument */
struct RE_string_args {
    RE_rpc_objID RPCobjID;
    string name<>;
};

/* structure for GetHostPlatformType results */
struct RE_get_host_platform_type_result {
    RE_rpc_objID RPCobjID;
    status;
    RE_erno_ty ptype;
};

/* structures for input and output of submit RPCs
 */
struct RE_submit_args {
    RE_rpc_objID RPCobjID;
    string hostname<;
    string directory<;
    int overwritePolicy;
    bool inplace;
    int transport;
    int submitObjectID;
    int socketPort;
    string socketClientName<;
    string mapFile_env<;
};

struct RE_get_submit_results_args {
    RE_rpc_objID RPCobjID;
    bool interrupt;
};

struct RE_get_submit_results_output {
    RE_rpc_objID RPCobjID;
    status;
    RE_erno_ty submitObjectID;
    int restore_engine.x3

```

```

    u_long objectsDone; /* handle for submit object */
};

/* structures for input of start RPC
 */
struct RE_start_args {
    RE_rpc_objID RPCobjID;
    submitObjectID; /* handle for submit object */
};

/* structures for input and output of get_restore_feedback RPC
 */
struct RE_get_restore_feedback_args {
    RE_rpc_objID RPCobjID;
    bool quit_restore; /* flag to request cancel */
};

struct RE_Notification {
    int msgType;
    int sourceModule;
    int level;
    int msgLen;
    string msgText<;
    RE_Notification *next;
};

struct RE_get_restore_feedback_result {
    RE_rpc_objID RPCobjID;
    RE_erno_ty status;
    EDMStats rstStats;
    RE_Notification *notify;
};

/* structure for output of get_question RPC
 */
struct RE_get_question_result {
    RE_rpc_objID RPCobjID;
    RE_erno_ty status;
    Question *query;
};

/* structure for input of set_user_answer RPC
 */
struct RE_set_user_answer_args {
    RE_rpc_objID RPCobjID;
    AnswerList answers;
};

/* structures for input and output of get_top_level_objects RPC
 */
struct RE_get_top_level_objects_args {
    RE_rpc_objID RPCobjID;
    string sourceHost<;
    short maxEntries;
    long cookie;
};

struct RE_get_top_level_objects_result {
    RE_rpc_objID RPCobjID;
    RE_erno_ty status;
    RSTRPC_tio_list *topLevelObjs; /* linked list */
    short numEntries;
    long cookie;
};

```

```

/* structures for input and output of get_workitem_templates rpc call:
*/
struct RE_get_top_level_templates_args {
    RE_rpc_objID    RPCobjID;
    RSTRPC_top_level_obj *topLevelObj;
    short           maxEntries;
    long            cookie;
};

struct RE_get_top_level_templates_result {
    RE_rpc_objID    RPCobjID;
    RE_erno_ty      status;
    short           numEntries;
    long            cookie;
    RSTRPC_name_list *templates; /* link to first template */
};

/* structure for input of does_alternate_exist rpc call:
*/
struct RE_does_alternate_exist_args {
    RE_rpc_objID    RPCobjID;
    RSTRPC_top_level_obj *topLevelObj;
    string           templateName<>;
};

/* structures for input and output of get_restorable_objects RPC's:
*/
struct RE_get_restorable_objects_start_args {
    RE_rpc_objID    RPCobjID;
    RE_erno_ty      status;
};

struct RE_get_restorable_objects_output_args {
    RE_rpc_objID;
    short           maxEntries;
};

struct RE_get_restorable_objects_output_result {
    RE_rpc_objID;
    RE_erno_ty      status;
    RSTRPC_uro_list *childrenObjs; /* linked list */
    long            numEntries;
    long            cookie;
};

/* structures for input and output of find_restorable_objects RPC's:
*/
struct RE_search_criteria {
    string startDirectory<256>; /* Dir to start searching */
    bool   descendDirectory; /* Flag to descend into subdirs */
    string searchString<128>; /* String to search for */
    bool   excludeString; /* Flag to include or exclude */
    RSTRPC_enum_ty typeOfFile; /* Types of files to search for */
    string owner<64>; /* Specific owner of files */
    bool   excludeOwner; /* Flag to exclude owner */
    string group<64>; /* Specific group of files */
};

```

```

bool   excludeGroup; /* Flag to exclude group */
RSTRPC_u_hyper sizeInBytes; /* Specific size of files to find */
RSTRPC_enum_ty sizeMatch; /* type of matching to do for size */
RSTRPC_time_ty startTime; /* First backup date to use */
RSTRPC_time_ty endTime; /* Last backup date to use */
};

struct RE_find_restorable_objects_args {
    RE_rpc_objID    RPCobjID;
    RE_search_criteria *searchCriteria;
};

struct RE_find_restorable_objects_result {
    RE_rpc_objID    RPCobjID;
    RE_erno_ty      status;
};

struct RE_get_find_results_args {
    RE_rpc_objID    RPCobjID;
    bool            interrupt;
    short           maxEntries;
    long            cookie;
};

struct RE_get_find_results_result {
    RE_rpc_objID    RPCobjID;
    RE_erno_ty      status;
    RSTRPC_found_obj_list *foundObjs; /* linked list */
    long            numEntries;
    long            cookie;
};

/* structures for input and output of mark_object RPC's:
*/
struct RE_mark_object_args {
    RE_rpc_objID    RPCobjID;
    RSTRPC_user_restorable_object *thisObj;
    RSTRPC_time_ty backupTime;
    bool           allowBadFiles;
    bool           descend;
};

struct RE_mark_object_result {
    RE_rpc_objID    RPCobjID;
    RE_erno_ty      status;
};

struct RE_get_mark_results_args {
    RE_rpc_objID    RPCobjID;
    bool            interrupt;
};

struct RE_get_mark_results_result {
    RE_rpc_objID    RPCobjID;
    RE_erno_ty      status;
    badFileCount;
    perbdenyFileCount;
    dirMarkCount;
    fileMarkCount;
    otherMarkCount;
};

/* structures for input and output of unmark_object RPC's:
*/

```

```

*/
struct RE_unmark_object_args {
    RE_rpc_objID      RPCobjID;
    RSTRPC_user_restorable_object *thisObj;
    RSTRPC_time_ty    backupTime;
    bool              badFilesOnly;
    bool              descend;
};

struct RE_get_unmark_results_result {
    RE_rpc_objID      RPCobjID;
    RE_erno_ty        status;
    u_long            badFileCount;
    u_long            dirMarkCount;
    u_long            fileMarkCount;
    u_long            otherMarkCount;
};

/* structure for output of get_marked_total_size RPC:
*/
struct RE_get_marked_total_size_result {
    RE_rpc_objID      RPCobjID;
    RE_erno_ty        status;
    RSTRPC_u_hypers    total;
};

/* structure for output of get_current_template RPC:
*/
struct RE_get_current_template_result {
    RE_rpc_objID      RPCobjID;
    RE_erno_ty        status;
    string             templateName<>;
    RSTRPC_bool        alternate;
};

/* structure for output of get_current_backup_time RPC:
*/
struct RE_get_current_backup_time_result {
    RE_rpc_objID      RPCobjID;
    RE_erno_ty        status;
    RSTRPC_time_ty    backupTime;
};

/* structure for input and output of get_all_backup_times RPC:
*/
struct RE_get_all_backup_times_args {
    RE_rpc_objID      RPCobjID;
    RSTRPC_time_ty    startTime;
    RSTRPC_time_ty    endTime;
    RSTRPC_backup_flags_ty flags;
    long              maxEntries;
    long              cookie;
};

struct RE_get_all_backup_times_result {
    RE_rpc_objID      RPCobjID;
    RE_erno_ty        status;
    RSTRPC_time_list *backupTimes;
    long              numEntries;
    long              cookie;
};

/* structure for input of is_there_xxxx_backup_for_time and
set_backup_for_time
*/
RPC's:
*/
Mon Oct 13 15:45:19 2008      restore_engine.x 7      Page 7 of 12

/* structure for input and output of is_object_markable RPC:
*/
struct RE_is_object_markable_args {
    RE_rpc_objID      RPCobjID;
    RSTRPC_time_ty    time;
    RSTRPC_backup_flags_ty flags;
};

/* structure for input of set_relative/backup * RPC's: */
struct RE_set_backup_time_args {
    RE_rpc_objID      RPCobjID;
    RSTRPC_backup_flags_ty flags;
};

/* structure for input and output of get_necessary_media RPC:
*/
struct RE_get_necessary_media_args {
    RE_rpc_objID      RPCobjID;
    long              maxEntries;
    RSTRPC_bool        all;
    long              cookie;
};

struct RE_get_necessary_media_result {
    RE_rpc_objID      RPCobjID;
    RE_erno_ty        status;
    RSTRPC_media_list *mediaList;
    short              numEntries;
    long              cookie;
};

/* structures for input and output of is_object_markable RPC:
*/
struct RE_is_object_markable_result {
    RE_rpc_objID      RPCobjID;
    RE_erno_ty        status;
    bool              markable;
};

/* structures for input and output of is_object_marked RPC:
*/
struct RE_is_object_marked_args {
    RE_rpc_objID      RPCobjID;
    RSTRPC_uro_list *objList;
    u_long              numEntries;
};

struct RE_is_object_marked_result {
    RE_rpc_objID      RPCobjID;
    RE_erno_ty        status;
    u_long              numMarked;
    RSTRPC_bool        marked<>;
};

/* structures for input and output of is_object_searchable and
* get_backup_times_support RPCs:
*/
struct RE_tlo_query_args {
    RE_rpc_objID      RPCobjID;
    RSTRPC_top_level_obj *topLevelObj;
};

struct RE_catalog_info {
    restore_engine.x 8

```

```

    RE_rpc_objID      RPCobjID;
    RE_errno_ty       status;
    string             level<>;
    string             numrec<>;
    string             catryec<>;
};

/* structure for inputs that require only time */
struct RE_time{
    RE_rpc_objID      RPCobjID;
    RE_errno_ty       status;
    RSTRPC_time_ty     backupTime;
};

struct RE_recx_file_info{
    RE_rpc_objID      RPCobjID;
    RE_errno_ty       status;
    RSTRPC_recx_file_info fileInfo;
};

program EDM_RESTORE_ENGINE {
    version EDMRE_FUNCTIONS {

        /* rpc for EDMRST_Initialize */
        RE_status_result
        re_initialize( RE_initialize_args ) = 1;

        /* rpc for EDMRST_GetSourceHosts */
        RE_get_hosts_result
        re_get_source_hosts( RE_get_hosts_args ) = 2;

        /* rpc for EDMRST_GetTopLevelObjects */
        RE_get_top_level_objects_result
        re_get_top_level_objects( RE_get_top_level_objects_args ) = 3;

        /* rpc for EDMRST_GetTopLevelTemplates */
        RE_get_top_level_templates_result
        re_get_top_level_templates(
            RE_get_top_level_templates_args ) = 4;

        /* rpc for EDMRST_Submit */
        RE_status_result
        re_submit( RE_submit_args ) = 5;

        /* rpc for EDMRST_GetSubmitResults */
        RE_get_submit_results_output
        re_get_submit_results( RE_get_submit_results_args ) = 6;

        /* rpc for EDMRST_Start */
        RE_status_result
        re_start( RE_start_args ) = 7;

        /* rpc for EDMRST_GetRestoreFeedback */
        RE_get_restore_feedback_result
        re_get_restore_feedback( RE_get_restore_feedback_args ) = 8;

        /* rpc for EDMRST_GetQuestion */
        RE_get_question_result
        re_get_question( RE_null_args ) = 9;

        /* rpc for EDMRST_SetUserAnswer */
        RE_status_result
        re_set_user_answer( RE_set_user_answer_args ) = 10;
    };
};

```

```

    re_set_user_answer( RE_set_user_answer_args ) = 10;

    /* rpc for EDMRST_Finish */
    RE_status_result
    re_finish( RE_null_args ) = 11;

    /* rpc for EDMRST_DoesAlternateExist */
    RE_boolean_result
    re_does_alternate_exist( RE_does_alternate_exist_args ) = 12;

    /* rpc's for EDMRST_GetRestorableObjects */
    RE_get_restorable_objects_start_result
    re_get_restorable_objects_start(
        RE_get_restorable_objects_start_args ) = 13;

    RE_get_restorable_objects_output_result
    re_get_restorable_objects_output(
        RE_get_restorable_objects_output_args ) = 14;

    /* rpc's for EDMRST_FindRestorableObjects */
    RE_find_restorable_objects_result
    re_find_restorable_objects(
        RE_find_restorable_objects_args ) = 15;

    RE_get_find_results_result
    re_get_find_results( RE_get_find_results_args ) = 16;

    /* rpc's for EDMRST_MarkObject */
    RE_mark_object_result
    re_mark_object( RE_mark_object_args ) = 17;

    RE_get_mark_results_result
    re_get_mark_results( RE_get_mark_results_args ) = 18;

    /* rpc's for EDMRST_UnmarkObject */
    RE_mark_object_result
    re_unmark_object( RE_unmark_object_args ) = 19;

    RE_get_unmark_results_result
    re_get_unmark_results( RE_get_unmark_results_args ) = 20;

    /* rpc for EDMRST_GetMarkedTotalSize */
    RE_get_marked_total_size_result
    re_get_marked_total_size( RE_null_args ) = 21;

    /* rpc for EDMRST_GetCurrentTemplate */
    RE_get_current_template_result
    re_get_current_template( RE_null_args ) = 22;

    /* rpc for EDMRST_GetCurrentBackupTime */
    RE_get_current_backup_time_result
    re_get_current_backup_time( RE_null_args ) = 23;

    /* rpc for EDMRST_GetAllBackupTimes */
    RE_get_all_backup_times_result
    re_get_all_backup_times( RE_get_all_backup_times_args ) = 24;

    /* rpc for EDMRST_IsTherePrevBackup */
    RE_boolean_result
    re_is_there_prev_backup( RE_set_backup_time_args ) = 25;

    /* rpc for EDMRST_IsThereNextBackup */
    RE_boolean_result
    re_is_there_next_backup( RE_set_backup_time_args ) = 26;

    /* rpc for EDMRST_IsTherePrevBackupForTime */
    RE_boolean_result
    re_is_there_prev_backup_for_time(
        RE_backup_for_time_args ) = 27;

    /* rpc for EDMRST_IsThereNextBackupForTime */
    RE_boolean_result
    re_is_there_next_backup_for_time(
        RE_backup_for_time_args ) = 28;
};

```

```

    RE_boolean_result
    re_is_there_next_backup_for_time(
        RE_backup_for_time_args ) = 28;

/* rpc for EDMRST_SetBackupForTime */
RE_status_result
re_set_backup_for_time( RE_backup_for_time_args ) = 29;

/* rpc for EDMRST_SetPrevBackup */
RE_status_result
re_set_prev_backup( RE_set_backup_time_args ) = 30;

/* rpc for EDMRST_SetNextBackup */
RE_status_result
re_set_next_backup( RE_set_backup_time_args ) = 31;

/* rpc for EDMRST_SetFirstBackup */
RE_status_result
re_set_first_backup( RE_set_backup_time_args ) = 32;

/* rpc for EDMRST_SetMostRecentBackup */
RE_status_result
re_set_most_recent_backup( RE_set_backup_time_args ) = 33;

/* rpc for EDMRST_GetNecessaryMedia */
RE_get_necessary_media_result
re_get_necessary_media( RE_get_necessary_media_args ) = 34;

/* rpc for EDMRST_IsObjectMarkable */
RE_is_object_markable_result
re_is_object_markable( RE_is_object_markable_args ) = 35;

/* rpc for EDMRST_IsObjectMarked */
RE_is_object_marked_result
re_is_object_marked( RE_is_object_marked_args ) = 36;

/* rpc for EDMRST_GetDestinationHosts */
RE_get_hosts_result
re_get_destination_hosts( RE_get_hosts_args ) = 37;

/* rpc for EDMRST_GetHostPlatformType */
RE_get_host_platform_type_result
re_get_host_platform_type( RE_string_args ) = 38;

/* rpc for EDMRST_IsObjectSearchable */
RE_boolean_result
re_is_object_searchable( RE_tlo_query_args ) = 39;

/* rpc for EDMRST_GetBackupTimesSupport */
RE_boolean_result
re_get_backup_times_support( RE_tlo_query_args ) = 40;

/* rpc for EDMRE_Load_recx_directives */
RE_status_result
re_load_recx_directives( RE_recx_file_info ) = 41;

/* rpc for EDMRST_poll_Load_recx_directives */
RE_status_result
re_poll_load_recx_directives( RE_null_args ) = 42;

/* rpc for RSTSL_get_backup_level */
RE_catalog_info
re_get_catalog_info( RE_time ) = 43;

/* rpc for EDMRST_GetAllTopLevelObjects */
RE_get_top_level_objects_result
re_get_top_level_objects(

```

```

    re_get_all_top_level_objects(
        RE_get_top_level_objects_args ) = 44;

/* rpc for EDMRST_GetSymmRestoreOption */
RE_boolean_result
re_get_symm_restore_option( RE_tlo_query_args ) = 45;

/* rpc for EDMRST_Ping */
RE_status_result
re_ping( RE_null_args ) = 46;

} = 1; /* This is version 1 */

/* This is the RPC program number.
   These are reserved in /pds/docs/RPC_numbers
   % * This number cannot be re-used by any other RPC daemon on the machine,
   % * identifies this daemon uniquely. If it were to be re-used,
   % * to register would be contacted when RPC's come in for this number.
   % */
} = 390016;

```

D4

EDMRST_GetAllTopLevelObjects	5	(RSTgettlob.c)
EDMRST_GetTopLevelObjects...	3	(RSTgettlob.c)

Wed Oct 08 16:05:30 2008	Page 1 of 6	Page 2 of 6
<pre> 2 /***** 3 ** File Name: RSTgettlb.c 4 ** 5 ** Copyright (c) 1998,1999 by EMC Corporation. 6 ** 7 ** Purpose: 8 ** This module contains the EDMRST_GetTopLevelObjects 9 ** Restore API function. 10 ** This function is provided to allow retrieval of the 11 ** top level objects which are restorable for the given client. 12 ** 13 ** 14 ** Compile-Time Options: 15 ** This section must list any compile time definitions 16 ** which will affect this header. 17 ** 18 *****/ 19 22 /* The following provides an RCS id in the binary that can be located 23 ** with the what(1) utility. The intent is to keep this short. 24 **/ 26 #ifndef lint 27 static char RCS_id [] = "\$RCSfile\$ " 28 "\$Revision\$ " 29 "\$Date\$"; 30 #endif 33 34 /* Feature test switches. 35 * Standard defines required to turn on OS features go here. 36 * 37 * The following is required for code that uses POSIX API's. 38 * Remove for non-POSIX, non-portable code. 39 */ 41 #define _POSIX_SOURCE 1 44 45 /* System headers. 46 */ 49 50 /* Epoch headers. 51 */ 52 #include <eb/eb_port.h> 53 #include <eb/rb_log.h> 56 57 /* Local headers 58 */ 59 #include <RSTinterns.h> 60 #include <RSTsup_rpc.h> 61 #include <RSTsup_csm.h> 64 65 * External declarations 66 */ </pre>	<pre> ***** ***** </pre>	<pre> ***** ***** </pre>
Wed Oct 08 16:05:30 2008	RSTgettlb.c 1	RSTgettlb.c 2
Wed Oct 08 16:05:30 2008	Page 1 of 6	Page 2 of 6

Wed Oct 08 16:05:30 2008		EDMRST_GetAllTopLevelObjects	Page 6 of 6
233	2	if (
234	2	RESTORABLE_OBJECT != objPtrArray[index]->restoreObjType	
235	2	NULL != objPtrArray[index]->rpcObjPtr)	
236	1	return(EP_RB_RECOVER_BAD_ARGS);	
237	1	}	
238	1	/* Prepare input argument structure for RPC: */	
239	1	rpc_args.sourceHost = (char *)sourceHost;	
240	1	rpc_args.maxEntries = maxEntries;	
241	1	rpc_args.cookie = *cookie;	
242	1	set_rpc_obj(
243	1	re_get_all_top_level_objects, &rpc_args.RPCobjID);	
244	1	rpc_result = re_get_all_top_level_objects.1(
245	2	if (!rpc_result) {	
246	2	rec_api_log_csm(SUB_CSM_RPC_FAIL, NULL);	
247	1	return(EP_RB_RECOVER_RPC_FAIL);	
248	1	}	
249	1	/* move results to caller's area, if successful: */	
250	1	if (rpc_result->status == E_SUCCESS)	
251	2	{	
252	2	*cookie = rpc_result->cookie;	
253	2	*numberEntries = rpc_result->numEntries;	
254	2	index = 0;	
255	2	while (rpc_result->numEntries)	
256	3	{	
257	3	temp_list = rpc_result->topLevelObjs;	
258	3	if (!temp_list !rpc_args.maxEntries--)	
259	3	/* some null pointer or too many	
260	3	returned */	
261	3	break;	
262	3	objPtrArray[index++] -> rpcObjPtr	
263	3	= {	
264	3	RSTRPC_restorable_obj_root *)temp_list->tlo;	
265	3	/* need this to end with NULL in	
266	3	rpc_result->topLevelObjs,	
267	3	* because returned top level objects can't be freed	
268	2	yet */	
269	2	rpc_result->topLevelObjs = temp_list->next;	
270	2	free(temp_list);	
271	1	rpc_result->numEntries--;	
272	1	}	
273	1	if (rpc_result->numEntries)	
274	1	rpc_result->status = EP_RB_RECOVER_SERVERFAIL;	
275	1	result = rpc_result->status;	
276	1	/* release RPC result struct: */	
277	1	xdr_free(xdr_re_get_top_level_objects_result, (
278	1	char *)rpc_result);	
279	1	return(result);	
280	1	/* end of EDMRST_GetNetworkTopLevelObjects() */	
281	1	}	
Wed Oct 08 16:05:30 2008		FSTgetlob.c 6	Page 6 of 6

Wed Oct 08 16:05:30 2008		EDMRST_GetAllTopLevelObjects	Page 5 of 6
176		/* ***** EDMRST_GetAllTopLevelObjects: ***** */	
177		* This function is provided to allow retrieval of the	
178		* work items which are restorable for the given client.	
179		* It is a GOAL of this routine to return all work-items ever backed	
180		* up successfully. Currently, though, it only looks in the config	
181		* file for work-items of the given client.	
182		* The cookie must be initialized to INIT_COOKIE on the first call to	
183		* this routine.	
184		* This routine will update the cookie to allow retrieval of more	
185		* objects if there are more than "maxEntries". The cookie will be	
186		* returned as DONE_COOKIE when there are no more to retrieve.	
187		* Parameters:	
188		* svrHdl (I) - A pointer to this user's client handle for the	
189		* Restore Engine (server) connection.	
190		* sourceHost (I) - the name of the source host being restored	
191		* maxEntries (I) - the maximum number of objects to return	
192		* topLevObjs (O) - ptr to pre-allocated array of restorableObject	
193		* buffer ptrs	
194		* numberEntries (O) - the real number of objects returned in the array	
195		* cookie (IO) - a place holder for the list position	
196		* meaningful to only the internals of the API	
197		***** */	
198		edmno_ty	
199		EDMRST_GetAllTopLevelObjects(serverHandle svrHdl,	
200		const char *sourceHost,	
201		const short maxEntries,	
202		restorableObjectPtr *topLevObjs,	
203		short *numberEntries,	
204		long *cookie)	
205		{	
206		RE_get_top_level_objects_result *rpc_result;	
207		RE_get_top_level_objects_args rpc_args;	
208		RSTRPC_tlo_list *temp_list;	
209		edmno_ty result = E_SUCCESS ;	
210		short index;	
211		restorableObject **objPtrArray = (
212		restorableObject **)topLevObjs;	
213		rbe_log_debug_sub(0, "EDMRST_GetAllTopLevelObjects called");	
214		/* validate args first: */	
215		if (sourceHost==NULL	
216		numberEntries==NULL	
217		cookie==NULL	
218		maxEntries <= 0	
219		topLevObjs== NULL	
220		svrHdl==NULL)	
221		return(EP_RB_RECOVER_BAD_ARGS);	
222		/* validate target restorableObjects: */	
223		for (index=0; index<maxEntries; index++)	
224		{	
225			
226			
227			
228			
229			
230			
231			
232			
Wed Oct 08 16:05:30 2008		RSTgetlib.c 5	Page 5 of 6

RSTSL_GetTopLevelObjects 3 (RSLgetblob.c)

Fri Oct 10 11:48:36 2008	Page 1 of 10	Page 2 of 10
<pre> 2 /***** 3 ** File Name: RSLgettlb.c 4 ** 5 ** Copyright (c) 1998,1999 by EMC Corporation. 6 ** 7 ** Purpose: 8 ** This module contains the RSTSL_GetTopLevelObjects 9 ** Restore Service Library function. 10 ** This function is provided to allow retrieval of the 11 ** top level objects which are restorable for the given client. 12 ** 13 ** 14 ** 15 ** Compile-Time Options: 16 ** This section must list any compile time definitions 17 ** which will affect this header. 18 ** 19 *****/ 22 23 /* The following provides an RCS id in the binary that can be located 24 ** with the what(1) utility. The intent is to keep this short. 25 */ 26 #ifndef lint 27 static char RCS_id [] = "SRCfiles\$ " 28 "\$Revision\$ " 29 "\$Date\$"; 30 #endif 31 32 33 /* Feature test switches. 34 * Standard defines required to turn on OS features go here. 35 * 36 * The following is required for code that uses POSIX API's. 37 * Remove for non-POSIX, non-portable code. 38 */ 39 41 #define _POSIX_SOURCE 1 42 43 44 /* System headers. 45 */ 46 47 48 /* Epoch headers. 49 */ 50 51 #include <eb/eb_port.h> 52 #include <eb/rb_log.h> 53 54 55 /* Local headers 56 */ 57 #include <RSLinterns.h> 58 59 60 /* #defines, structures, typedefs local to this source file 61 */ 62 </pre>	<pre> 66 typedef struct wi_info 67 { 68 struct wi_info *next; 69 char *wi_name; 70 char *wi_work; 71 char *wi_bic; 72 char wi_type; 73 } wi_info; 74 75 NEW_SRC_FILE(); 76 77 /* External declarations 78 */ 79 </pre>	<pre> 66 typedef struct wi_info 67 { 68 struct wi_info *next; 69 char *wi_name; 70 char *wi_work; 71 char *wi_bic; 72 char wi_type; 73 } wi_info; 74 75 NEW_SRC_FILE(); 76 77 /* External declarations 78 */ 79 </pre>
Fri Oct 10 11:48:36 2008	Page 1 of 10	Page 2 of 10

Fri Oct 10 11:48:36 2008	RSTSL_GetTopLevelObjects	Page 3 of 10
<pre> 144 1 144 1 144 1 145 1 145 1 145 1 147 1 147 1 147 1 148 2 148 2 148 2 149 2 149 2 149 2 150 1 150 1 150 1 152 1 152 1 152 1 153 1 153 1 153 1 154 1 154 1 154 1 155 1 155 1 155 1 156 1 156 1 156 1 158 1 158 1 158 1 159 2 159 2 159 2 160 2 160 2 160 2 161 1 161 1 161 1 163 1 163 1 163 1 164 1 164 1 164 1 165 1 165 1 165 1 166 1 166 1 166 1 168 1 168 1 168 1 169 2 169 2 169 2 170 2 170 2 170 2 171 1 171 1 171 1 173 1 173 1 173 1 174 2 174 2 174 2 175 2 175 2 175 2 176 2 176 2 176 2 177 3 177 3 177 3 178 3 178 3 178 3 179 3 179 3 179 3 180 2 180 2 180 2 181 1 181 1 181 1 182 1 182 1 182 1 183 2 183 2 183 2 184 2 184 2 184 2 185 1 185 1 185 1 187 1 187 1 187 1 189 1 189 1 189 1 190 2 190 2 190 2 191 2 191 2 191 2 192 2 192 2 192 2 193 2 193 2 193 2 195 2 195 2 195 2 196 2 196 2 196 2 197 2 197 2 197 2 199 2 199 2 199 2 200 2 200 2 200 2 201 2 201 2 201 2 202 2 202 2 202 2 203 2 203 2 203 2 204 2 204 2 204 2 </pre>	<pre> /* If the pointer is NULL, return error. */ if (NULL == rcp->rc_config) { return(EP_RB_RECOVER_BAD_CONTEXT); } /* * Check value of cookie. If it is the init cookie, then call * rb_get_auth_clients() otherwise, feed off of list of names that * were generated by the first call. */ if (NULL == cookie) { return(EP_RB_RECOVER_BAD_COOKIE); } /* * Fill in the recover context source client hostname field * If it currently is set to something, free up that memory first. */ if (NULL != rcp->rc_source_client_hostname) { free(rcp->rc_source_client_hostname); } if (sourceHost && *sourceHost != 0) { rcp->rc_source_client_hostname = esl_strdup(sourceHost); if (NULL == rcp->rc_source_client_hostname) { rec_api_log_csm(SUB_CSM_NOMEM, NULL); return(EP_RB_RECOVER_NOMEM); } } else { return EP_RB_RECOVER_BAD_ARGS; } rbe_log_debug(0, "In GetTLO for %s, cookie = %x", sourceHost, *cookie); if (INIT_COOKIE == *cookie) { /* * Set initial values since the cookie indicates the INIT state. */ wi_count = 0; pi_count = 0; wi_list = NULL; /* * Check to see if there is already an in-memory static list of * workitems being used by this function. If so, free this list * since the caller has reset the cookie to INIT without having * exhausted the list. */ } </pre>	Page 4 of 10
Fri Oct 10 11:48:36 2008	RSTSL_GetTopLevelObjects	Page 3 of 10
<pre> 82 1 82 1 82 1 83 1 83 1 83 1 84 1 84 1 84 1 85 1 85 1 85 1 86 1 86 1 86 1 87 1 87 1 87 1 88 1 88 1 88 1 89 1 89 1 89 1 90 1 90 1 90 1 91 1 91 1 91 1 92 1 92 1 92 1 93 1 93 1 93 1 94 1 94 1 94 1 95 1 95 1 95 1 96 1 96 1 96 1 97 1 97 1 97 1 98 1 98 1 98 1 99 1 99 1 99 1 100 1 100 1 100 1 101 1 101 1 101 1 102 1 102 1 102 1 103 1 103 1 103 1 104 1 104 1 104 1 105 1 105 1 105 1 </pre>	<pre> /* * RSTSL_GetTopLevelObjects: * * This function is provided to allow retrieval of the * work items which are restorable for the given client. * * It is a GOAL of this routine to return all work-items ever backed * up successfully. Currently, though, it only looks in the config * file for network workitems of the given client. * * The cookie must be initialized to INIT_COOKIE on the first call to * this * routine. * * This routine will update the cookie to allow retrieval of more * objects if there are more than "maxEntries". The cookie will be * returned as DONE_COOKIE when there are no more to retrieve. * * Parameters: * sourceHost (I) - the name of the source host being restored * maxEntries (I) - the maximum number of objects to return * topLevObjs (O) - ptr to linked list of Top Level Objects * numberEntries (O) - the real number of objects returned in the array * cookie (IO) - a place holder for the list position * meaningful to only the internals of the API */ eerrno_ty RSTSL_GetTopLevelObjects(const char *sourceHost, const short maxEntries, struct RSTRPC_tlo_list **topLevObjs, short *numberEntries, long *cookie, int execMode) { RBC_WORKGROUP *wgp; short index; wi_info *n_node; wi_info *tmp_list; short tmp_count; eerrno_ty result = E_SUCCESS; void *saved_appdata; struct RSTRPC_tlo_list *tloPtr; struct RSTRPC_tlo_list *tloLast; struct pluginData *piPtr; /* * Note that the following variables are declared static so that * values set during the initial call to this routine will be * of use on subsequent calls when no explicit set takes place. */ static static wi_count; static total_count; static valid_cookie; static *wi_list; static wi_info *top_wi_list=NULL; static pi_count; struct RSTRPC_tlo_list *pluginList = NULL; } /* * Check to make sure rcp has the in memory config info. */ </pre>	Page 3 of 10
Fri Oct 10 11:48:36 2008	RSLgettlb.c	Page 4 of 10
<pre> 142 1 142 1 142 1 143 1 143 1 143 1 </pre>	<pre> /* * Check to make sure rcp has the in memory config info. */ </pre>	Page 4 of 10

Fri Oct 10 11:48:36 2008			RSTSL_GetTopLevelObjects			RSTSL_GetTopLevelObjects			Page 5 of 10		
206	2		while (NULL != top_wi_list)			266	5		/* For striped workitems, the work item list can be		
207	3		{			267	5		* NULL for the stripes > 1 of workitems without a		
208	3		{			268	5		* partitionspec.		
209	4		{			269	5		*/		
210	4		free(top_wi_list->wi_name);			270	5				
211	3		}			271	5				
212	3		if (NULL != top_wi_list->wi_work)						NULL != wip->list) /* else wi_work already NULL */		
213	4		{			272	6		{		
214	4		{			273	6		n_node->wi_work = esl_strdup(wip->list);		
215	3		free(top_wi_list->wi_work);			274	6		if (NULL == n_node->wi_work)		
216	3		{			275	7		{		
217	4		if (NULL != top_wi_list->wi_bic)			276	7		result = EP_RB_RECOVER_NOMEM;		
218	4		{			277	7		break;		
219	3		free(top_wi_list->wi_bic);			278	6		}		
220	3		}			279	5				
221	3		tmp_list = top_wi_list->next;			281	5		if (NULL != wip->backup_init)		
222	3		free(top_wi_list);			282	6		{		
223	2		top_wi_list = tmp_list;			283	6		{		
			}			284	6		n_node->wi_bic = esl_strdup(wip->backup_init);		
225	2		/* Free unused top level objects from plugins */			285	7		if (NULL == n_node->wi_bic)		
226	2		if (pluginList)			286	7		{		
227	3		{			287	7		result = EP_RB_RECOVER_NOMEM;		
228	3		RSTSL_FreeRestorableObjectList(pluginList);			288	6		break;		
229	3		pluginList = NULL;			289	5		}		
230	2		}						n_node->wi_type = wip->wi_type;		
			/*			291	5		++wi_count;		
232	2		* Scan all the workitems in all the workgroups			293	5		}		
233	2		* We are going to get all of the items that meet the			294	4		} /* end inner for(
234	2		criteria			295	3) scanning workitems within a workgroup */		
			* and keep them in static memory. Further calls will feed off						if (
236	2		* of this list.			297	3		result != E_SUCCESS) /* malloc failure, break out */		
237	2		*/			298	3		break;		
			for (wgp = rcp->rc_config->pgrouplist;						} /* end outer for() scanning workgroups */		
239	2		wgp != NULL ;			300	2		total_count = 1; /* Needs to be 1 based */		
240	2		wgp = wgp->next)			302	2		valid_cookie = INIT_COOKIE; /* Clean up cookie crumbs */		
241	2		{			303	2		/*		
242	3		RBC_WORKITEM *wip;			305	2		* Save the pointer to the head of the list for freeing later.		
243	3					306	2		*/		
245	3		for (wip = wgp->pwilist; wip != NULL; wip = wip->next)						top_wi_list = wi_list;		
246	4		{			309	2		if (result != E_SUCCESS) {		
247	4		/* host must match and witype must not be a plug-in's			311	3		if (result == EP_RB_RECOVER_NOMEM) {		
			*/			312	4		rec_api_log_csm(SUB_CSM_NOMEM, NULL);		
248	4		if ((0 == strcmp(wip->sysname, sourceHost))			313	4		}		
249	4		&& ((execMode == RAW_NETWORK)			314	3		/* free wi_list next time in */		
250	4		(0 == rcp->rc_num_plugin_wi_types			315	3		return result;		
251	4		NULL == strchr(316	3				
			rcp->rc_plugin_wi_types, wip->wi_type,			317	2		if (execMode != RAW_NETWORK)		
252	4		rcp->rc_num_plugin_wi_types)						{		
)))			319	2		/* save appData pointer in case a tlo is selected already */		
253	5		{			320	3		saved_appdata = rcp->appdata;		
254	5		{			321	3				
			n_node = linked_list_new((322	3		/* get top level object list(s) from plugin(s) */		
255	6		generic_list_ty **) &wi_list, sizeof(wi_info));						for (index = 1, piPtr = rcp->piList;		
256	6		if (n_node == NULL) {			324	3		NULL != piPtr;		
257	6		result = EP_RB_RECOVER_NOMEM;			325	3		index++, piPtr = piPtr->next)		
258	5		break;			326	3				
259	5		n_node->wi_name = esl_strdup(wip->name);						/* get top level object list(s) from plugin(s) */		
260	5		if (NULL == n_node->wi_name)			327	3		for (index = 1, piPtr = rcp->piList;		
261	6		{						NULL != piPtr;		
262	6		{						index++, piPtr = piPtr->next)		
263	6		result = EP_RB_RECOVER_NOMEM;								
264	5		break;								
264	5										

Fri Oct 10 11:48:36 2008		RSTSL_GetTopLevelObjects		Page 7 of 10		
328	4	{		385	1	/*
329	4	{	tloPtr = NULL;	386	1	* Logic drops through to here regardless of the passed in cookie
330	4		tmp_count = 0;	387	1	* being the INIT state or a valid key in a subsequent call.
331	4		rcp->appData = piPtr->appData;	388	1	*/
332	4		result = piPtr->piFuncArray[PIFuncIndexGetTLO]			/*
333	4		{	390	1	* Return a linked list of the number of "restorable objects"
			rcp, sourceHost, &tloPtr, &tmp_count);	391	1	requested.
334	4		if (result != E_SUCCESS	392	1	* Stop if we reach the end of the list.
335	4		(NULL == tloPtr && tmp_count > 0)	393	1	*/
336	4		(NULL != tloPtr && tmp_count == 0))			/* set linked list and current entry pointers to NULL at start */
337	5		{ /* error or inconsistent results */	395	1	*topLevObjs = tloPtr = NULL;
338	5		rbe_internal_error(396	1	index = 0;
339	5		result,			while ((total_count <= wi_count + pi_count)
340	5		"plugin: %s;	398	1	&& (index < maxEntries)
			error in GetTopLevelObjects call",	399	1	&& ((wi_list != NULL) (pluginList != NULL)))
341	5		((struct pluginIDdata *) (401	1	{
342	5		piPtr->idData))->name	402	2	{
);	403	2	{
344	5		result = E_SUCCESS;	404	3	{
345	5		continue;	405	3	{
			/* try next plugin anyway */	406	3	{
346	4		}	407	3	{
347	4		if (NULL == tloPtr)	408	3	{
348	4		continue;	409	3	{
			/* none from this plug-in */	410	4	{
350	4		if (NULL == pluginList)	411	4	{
351	4		pluginList = tloPtr;	412	4	{
352	4		else			/* do we return partial list, or free list & return none ? */
353	4		tloLast->next = tloPtr;			/* need a local function to free list of restorable objects
			for (; tloPtr && tmp_count;			on error */
355	4		tmp_count--, pi_count++)	413	4	{
356	5		{	414	4	{
357	5		{	415	3	{
358	5		{	417	3	{
359	5		{	418	3	{
360	4		{	419	3	{
			if (420	4	{
362	4		tmp_count tloPtr)	421	4	{
363	5		{	422	4	{
364	5		{	423	3	{
365	5		{			tloPtr->tlo->root.backupApp = 0;
366	5		{			/* value for network objects */
			rbe_internal_error(427	3	{
			0,	428	4	{
			"plugin: %s;	429	4	{
			bad output from GetTopLevelObjects call",	430	4	{
			((struct pluginIDdata *) (431	5	{
			piPtr->idData))->name	432	5	{
);	433	5	{
			};	434	4	{
			};	435	3	{
372	3		rcp->appData = saved_appdata;			/* restore saved appData ptr */
			/* restore saved appData ptr */	373	2	{
			{	374	2	{
			{	375	3	{
			{	376	3	{
			{	377	3	{
			{	378	2	{
379	1		pluginList = NULL;			if (NULL != wi_list->wi_bic)
380	1		pi_count = 0;			{
			}			{
			}			tloPtr->tlo->wiBIC = esl_strdup(wi_list->wi_bic);
			}			if (NULL == tloPtr->tlo->wiBIC)
381	2		else if ((valid_cookie != *cookie) (DONE_COOKIE == *cookie))			{
382	2		{			{
383	1		return(EP_RB_RECOVER_BAD_COOKIE);			result = EP_RB_RECOVER_NOMEM;
			}			break;
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}			}
			}		</	

```

444 4      }
445 3      }
447 3      tloPtr->tlo->wiType = wi_list->wi_type;
449 3      wi_list = wi_list->next;
450 2
451 2      else /* get tlo from plugin list: */
452 3      {
453 3          if (NULL == *topLevObjjs)
454 4          {
455 4              /* set start of list ptr to plugin tlo's */
456 3              *topLevObjjs = pluginList;
457 3          }
458 4          else
459 4          {
460 3              /* link prev entry to this one in plugin list */
461 3              tloPtr->next = pluginList;
462 3          }
463 3          tloPtr = pluginList;
464 2          pluginList = pluginList->next;
465 2          tloPtr->next = NULL;
466 2      }
467 2      /* set hostname in all top level objects */
468 2      tloPtr->tlo->hostname = esl_strdup( sourceHost );
469 3      if (NULL == tloPtr->tlo->hostname)
470 3      {
471 3          result = EP_RB_RECOVER_NOMEM;
472 2          break;
473 2      }
474 2      ++total_count;
475 1      ++index;
476 1      }
477 1      /* catch all malloc failures here: free list, return failure (
nothing) */
478 2      if (result != E_SUCCESS) {
479 2          rec_api_log_csm(SUB_CSM_NOMEM, NULL);
480 2          index = 0;
481 2          valid_cookie = INIT_COOKIE; /* to indicate none returned */
482 2          total_count = wi_count + pi_count + 1; /* to indicate restart needed */
483 2          /* free linked list output */ /* to cause free of wi lists */
484 2          if (*topLevObjjs != NULL)
485 3          {
486 3              RSTSL_FreeRestorableObjectList( *topLevObjjs );
487 3              *topLevObjjs = NULL;
488 2          }
489 1      }
490 1      *numberEntries = index;
491 1
492 1      /* Set the cookie appropriately.
Also store it in the static "valid_cookie"
*/
493 1
494 1      if (wi_count + pi_count >= total_count) /* any more TLO's for
later? */
495 1      {
496 1          /* yes: */
497 1          if (wi_count >= total_count) { /* any more netwk TLOs? */
498 2              *cookie = (long)wi_list; /* yes, still have netwk wis */
499 3          }
500 3      }
501 2

```

```

502 3      else {
503 3          *cookie = (long)pluginList; /* no, rest are plugin TLOs */
504 2      }
505 2      valid_cookie = *cookie;
506 1      }
507 1      else
508 2      {
509 2          /* Set the cookie to DONE state and the valid_cookie to match
it.
*/
510 2          if (result == E_SUCCESS)
511 2          {
512 2              /* dont set cookie on errors */
513 3              *cookie = DONE_COOKIE;
514 3              valid_cookie = DONE_COOKIE;
515 3          }
516 3          else
517 2          {
518 2              /* Free up memory. We are done and don't need it anymore.
Start from the top of the list.
*/
519 2              while (NULL != top_wi_list)
520 2              {
521 2                  if (NULL != top_wi_list->wi_name)
522 3                  {
523 3                      free(top_wi_list->wi_name);
524 2                  }
525 2                  if (NULL != top_wi_list->wi_work)
526 3                  {
527 3                      free(top_wi_list->wi_work);
528 2                  }
529 2                  if (NULL != top_wi_list->wi_bic)
530 3                  {
531 3                      free(top_wi_list->wi_bic);
532 2                  }
533 2                  tmp_list = top_wi_list->next;
534 2                  free(top_wi_list);
535 2                  top_wi_list = tmp_list;
536 2              }
537 2          }
538 2          if (pluginList)
539 3          {
540 3              RSTSL_FreeRestorableObjectList( pluginList );
541 3              pluginList = NULL;
542 2          }
543 2          return( result );
544 1          /* end of RSTSL_GetTopLevelObjects() */
545 1      }
546 1
547 1
548 1
549 1
550 1
551 1

```

D5

EDMRST_GetRestorableObjects 3 (RSTgetrobs.c)

```
2  /*****
3  **
4  ** File Name:   RSTgetrobs.c
5  **
6  ** Copyright (c) 1998,1999 by EMC Corporation.
7  **
8  ** Purpose:
9  **
10 ** This module contains the EDMRST_GetRestorableObjects API and some
11 ** support functions.
12 **
13 ** Table of Contents:
14 ** -----
15 ** API Functions:
16 **
17 ** EDMRST_GetRestorableObjects
18 **
19 **
20 ** Compile-Time Options:
21 **
22 *****/
23
24
25 /* The following provides an RCS id in the binary that can be located
26 ** with the what(1) utility. The intent is to keep this short.
27 */
28
29 #ifndef lint
30 static char RCS_id [] = "$RCSfile$ "
31                      "$Revisions$ "
32                      "$Date$";
33 #endif
34
35 /*
36 ** Feature test switches.
37 ** Standard defines required to turn on OS features go here.
38 **
39 ** The following is required for code that uses POSIX API's.
40 ** Remove for non-POSIX, non-portable code.
41 **
42 */
43
44 #define _POSIX_SOURCE 1
45
46
47 /*
48 ** System headers.
49 **
50 #include <unistd.h> /* doesn't help define usleep(), so... */
51 extern int usleep( unsigned int );
52
53
54 /*
55 ** Epoch headers.
56 **
57 #include <eb/eb_port.h>
58 #include <eb/rb_log.h>
59
60 /*
61 ** Local headers
62 */
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
```

```
64 #include <RSTinterns.h>
65 #include <RSTsup_csm.h>
66
67 /*
68 ** #defines, structures, typedefs local to this source file
69 **
70 */
71
72 #define RST_MAX_GET_ROBJS_DELAY 3 /* max seconds between polls */
73
74 /*
75 ** External declarations
76 **
77 NEW_SRC_FILE();
78
79
80
81 /*
82 ** Local function prototypes
83 */
```

```

85 /*****
86 * EDMRST_GetRestorableObjects API
87 *
88 * Function Description:
89 *
90 * Given a parent (
91 *   i.e. container) restorable object, return the children
92 *   restorable objects it contains.
93 *
94 * The cookie must be initialize to INIT_COOKIE on the first call to
95 *   this routine.
96 * This routine will update the cookie to allow retrieval of more
97 * objects if there is more than "maxEntries". The cookie will be
98 * returned as DONE_COOKIE when there are no more to retrieve.
99 *
100 * Parameters:
101 *   svrHdl - (
102 *     I) A pointer to this user's client handle for the
103 *     Restore Engine (server) connection.
104 *   parentPtr - (I) ptr to parent restorableObject
105 *   allowBF - (
106 *     I) flag indicating whether or not to include bad
107 *     files
108 *   maxEntries - (
109 *     I) max. # of entries that the preallocated buffer
110 *     objBufPtr - (
111 *       I) ptr to preallocated array of restorableObject
112 *       numEntries - (
113 *         I) ptr to buffer to receive number of entries
114 *         returned in objBufArray
115 *         cookie - (
116 *           I/O) ptr to a long integer whose value is meaningful
117 *           to only the internals of the API
118 *
119 * Return Codes:
120 *   E_SUCCESS - operation completed successfully
121 *   EP_RB_RECOVER_INVALID_OBJNAME - input restorableObject does
122 *   not have a valid name;
123 *   EP_RB_RECOVER_BAD_ARGS - input restorableObject ptr
124 *   or objBufPtr is NULL;
125 *   EP_RB_RECOVER_BAD_COOKIE - input cookie ptr is NULL, or
126 *   the cookie is DONE_COOKIE;
127 *   EP_RB_RECOVER_INVALIDOP - the call is issued without
128 *   the correct context setup;
129 *   return codes from GetWorkItemContents() and from
130 *   GetDirContents().
131 */
132 gerno_ty
133 EDMRST_GetRestorableObjects( serverHandle svrHdl,
134 const restorableObjectPtr parentPtr,
135 const boolean_ty allowBF, maxEntries,
136 restorableObjectPtr *objBufPtr,
137 long *numEntries,
138 long *cookie )
139 {
140 RE_get_restorable_objects_start_result *start_rpc_result =
141 1

```

```

142 1 RE_get_restorable_objects_start_args start_rpc_args;
143 1 RE_get_restorable_objects_output_result output_rpc_result = NULL;
144 1 RE_get_restorable_objects_output_args output_rpc_args;
145 1 RSTRPC_uio_list *temp_list;
146 1 struct RSTRPC_restorable_obj_root *temp_robj;
147 1 gerno_ty result = E_SUCCESS;
148 1 short index;
149 1 restorableObject **objPtrArray = (
150 1   restorableObject **)objBufPtr;
151 1
152 1 rbe_log_debug_sub( 0, "EDMRST_GetRestorableObjects called" );
153 1
154 1 /* validate args first: */
155 1 if (parentPtr==NULL)
156 1 || numEntries==NULL
157 1 || cookie=NULL
158 1 || maxEntries <= 0
159 1 || objBufPtr==NULL
160 1 || svrHdl=NULL )
161 1 return( EP_RB_RECOVER_BAD_ARGS );
162 1
163 1 /* validate target restorableObjects: */
164 1 for ( index=0; index<maxEntries; index++ )
165 1 {
166 1   if ( ( NULL == objPtrArray[index] )
167 1       || (
168 1         RESTORABLE_OBJECT != objPtrArray[index]->restorableObjectType
169 1         || ( NULL != objPtrArray[index]->rpcobjPtr )
170 1         )
171 1       )
172 1     return( EP_RB_RECOVER_BAD_ARGS );
173 1
174 1   if (*cookie == DONE_COOKIE)
175 1     return(EP_RB_RECOVER_BAD_COOKIE);
176 1
177 1   /* validate parent object type as top level or container */
178 1   if ( NULL == (temp_robj = (
179 1     restorableObject *)parentPtr->rpcobjPtr))
180 1     return( EP_RB_RECOVER_BAD_ARGS );
181 1
182 1   if ( (RESTORABLE_OBJECT !=
183 1     (restorableObject *)parentPtr->restorableObjectType )
184 1     || return EP_RB_RECOVER_INVALID_OBJTYPE;
185 1
186 1   if ( (temp_robj->objlevel != RSTRPC_tlo_type)
187 1     && (temp_robj->objlevel != RSTRPC_container_type) )
188 1     if (temp_robj->objlevel != RSTRPC_leaf_type)
189 1       return( EP_RB_RECOVER_INVALID_OBJTYPE );
190 1     else
191 1       return( EP_RB_RECOVER_INVALIDOP );
192 1
193 1   /* request the restorable objects with one RPC */
194 1   /* Prepare input argument structure for RPC: */
195 1   struct RE_restorable_obj {
196 1     struct RE_restorable_obj *parentObj;
197 1     struct RE_restorable_obj *u.uiointo;
198 1     struct RSTRPC_user_restorable_object *temp_robj;
199 1     start_rpc_args.allowBadFiles = allowBF;
200 1     start_rpc_args.cookie = *cookie;
201 1     start_rpc_args.maxEntries = maxEntries;

```

```

202 1      set_rpc_obj( re_get_restorable_objects_start,
203 1          kstart_rpc_args.RPCobjID );
205 1      start_rpc_result = re_get_restorable_objects_start_1(
206 1          kstart_rpc_args,
207 1          svrHdl );
207 2      if (!start_rpc_result) {
208 2          result = EP_RB_RECOVER_RPC_FAIL;
209 2          rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL);
210 1      }
211 1      else
212 2      {
213 2          result = start_rpc_result->status;
215 2      }
216 2      /* release RPC result struct: contents and struct */
217 2      xdr_free( xdr_RE_get_restorable_objects_start_result,
218 1          (char *)start_rpc_result );
220 1      free( start_rpc_args.parentObj );
222 1      /* prepare to call another RPC for results, if successful: */
223 1      if (result != E_SUCCESS)
224 1          return( result );
225 1      else
226 1          result = EP_RB_RECOVER_RPC_INCOMPLETE;
228 1      output_rpc_args.maxEntries = maxEntries;
230 1      /* poll for completion or error */
231 1      while (result == EP_RB_RECOVER_RPC_INCOMPLETE)
232 2      {
233 2          unsigned int poll_delay = 100000; /* .1 second */
234 2          set_rpc_obj( re_get_restorable_objects_output,
235 2              koutput_rpc_args.RPCobjID );
236 2          output_rpc_result =
237 2              re_get_restorable_objects_output_1(
238 1                  koutput_rpc_args,
239 1                  svrHdl );
240 3          if (!output_rpc_result) {
241 3              result = EP_RB_RECOVER_RPC_FAIL;
242 2              rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL);
243 2          }
244 2          else
245 2              result = output_rpc_result->status;
246 2      }
247 3      if (result == EP_RB_RECOVER_RPC_INCOMPLETE)
248 3      {
249 3          /* release RPC result struct: contents and
250 3              struct */
251 3          xdr_free(
252 3              xdr_RE_get_restorable_objects_output_result,
253 3              (char *)output_rpc_result );
254 3          output_rpc_result = NULL;
255 3          /* wait till next poll */
256 3          usleep( poll_delay );
257 4          if (poll_delay < RST_MAX_GET_ROBJS_DELAY) {
258 4              poll_delay *= 2;
259 4              if (poll_delay > RST_MAX_GET_ROBJS_DELAY)
260 1                  poll_delay = RST_MAX_GET_ROBJS_DELAY;
260 1          }

```

```

262 1      /* move results to caller's area, if successful: */
263 1      if (result == E_SUCCESS)
264 2      {
265 2          *cookie = output_rpc_result->cookie;
266 2          *numEntries = output_rpc_result->numEntries;
267 2          index = 0;
268 2          while ( output_rpc_result->numEntries )
269 3          {
270 3              temp_list = output_rpc_result->childrenObjs;
271 3              if (
272 3                  temp_list || !output_rpc_args.maxEntries-- )
273 3                  break;
274 3              /* null pointer or too many returned */
275 3              objPtrArray[index]-->rpcObjPtr =
276 3                  RSTRPC_restorable_obj_root *temp_list->uro;
277 3              /* needed to end with NULL in
278 3                  output_rpc_result->childrenObjs,
279 3                  * because returned user rest.
280 3                  objects can't be freed yet */
281 2              output_rpc_result->childrenObjs =
282 2                  temp_list->next;
283 2              index++;
284 1          }
286 1          if (output_rpc_result->numEntries)
287 2              result = EP_RB_RECOVER_SERVERFAIL;
288 2      }
289 2      /* release RPC result struct's contents and itself: */
290 1      if (output_rpc_result) {
291 1          xdr_free(
292 1              xdr_RE_get_restorable_objects_output_result,
293 1              (char *)output_rpc_result );
293 1      }
293 1      return( result );
293 1      /* EDMRST_GetRestorableObjects */

```

--	--

--	--

GetTIOContents	6	(RSLgetrobs.c)
RSTSL_GetRestorableObjects...	3	(RSLgetrobs.c)

```
2  /*****
3  **
4  ** File Name:  RSLgetrobs.c
5  **
6  ** Copyright (c) 1998,1999 by EMC Corporation.
7  **
8  ** Purpose:
9  **
10 **      This module contains the RSTSL_GetRestorableObjects Restore
11 **      Library API.
12 **
13 ** Table of Contents:
14 ** -----
15 **
16 **      RSTSL_GetRestorableObjects
17 **
18 **      Internal Functions:
19 **
20 **      GetTIOContents
21 **
22 ** Compile-time Options:
23 **      This section must list any compile time definitions
24 **      which will affect this header.
25 **
26 *****/
27
28
29 /* The following provides an RCS id in the binary that can be located
30 ** with the what(1) utility. The intent is to keep this short.
31 **
32
33 #ifndef lint
34 static char RCS_id [] = "$RCSfile$ "
35                        "$Revision$ "
36                        "$Date$";
37 #endif
38
39
40 /*
41  * Feature test switches.
42  *
43  * Standard defines required to turn on OS features go here.
44  *
45  * The following is required for code that uses POSIX API's.
46  *
47  * Remove for non-POSIX, non-portable code.
48
49 #define _POSIX_SOURCE 1
50
51
52 /*
53  * System headers.
54
55
56 /*
57  * Epoch headers.
58 */
59 #include <eb/eb_port.h>
60 #include <eb/rb_log.h>
61 #include <ebutil/eb_locking.h>
```

```
64 /*
65  * Local headers
66 */
67 #include <RSLintern.h>
68 #include <RSLstartup.h>
69
70
71 /*
72  * #defines, structures, typedefs local to this source file
73 */
74
75
76 /*
77  * External declarations
78 */
79
80 NEW_SRC_FILE();
81
82
83 /*
84  * Local function prototypes
85 */
86
87 static eerrno_t GetTIOContents(
88     struct RSTRPC_top_level_obj *tloobjPtr,
89     const boolean_t allowBf,
90     const long maxEntries,
91     struct RSTRPC_uro_list **objListPtr,
92     long *cntEntries,
93     long *cookie);
```

```

94  /*****
95  * RSTSL_GetRestorableObjects:
96  *
97  * This function is provided to allow retrieval of the
98  * child objects which are restorable given the parent object. The
99  * caller specifies the parent object and whether or not
100  * to include bad files.
101  *
102  * Performs the 'get' on the asynchronous thread of the Restore
103  * Engine.
104  *
105  * Parameters:
106  *   parentPtr      (I) - the parent object: pointer to either a top level
107  *   objectLevel    (I) - specifies whether parentObject points to top level or
108  *   objects         (O) - a pre-allocated pointer to return the list of
109  *                     objects in
110  *   cookie          (IO) - a place holder for the list position
111  *   maxEntries     (I) - the maximum number of objects to return
112  *   cntEntries     (I) - the real number of objects returned in the list
113  *   allowBF        (I) - flag whether or not to include bad files
114  *
115  * ****
116  * eerrno_ty RSTSL_GetRestorableObjects (
117  *     restorableObjectPtr      parentPtr,
118  *     enum RSTRPC_ObjectLevel  objectLevel,
119  *     struct RSTRPC_uro_list    **objects,
120  *     long                     *cookie,
121  *     const long               *maxEntries,
122  *     long                     *cntEntries,
123  *     const boolean_ty         allowBF )
124  {
125  eerrno_ty result;
126  struct RSTRPC_uro_list *uroList;
127  struct RSTRPC_top_level_obj *tloPtr
128  = (struct RSTRPC_top_level_obj *)parentPtr;
129  struct RSTRPC_user_restorable_object *uroPtr
130  = (struct RSTRPC_user_restorable_object *)parentPtr;
131
132  /*
133  * Validate the input arguments.
134  */
135
136  if ((NULL == parentPtr) || (NULL == objects) || (
137      {
138          return(EP_RB_RECOVER_BAD_ARGS);
139      }
140      if (objectLevel == RSTRPC_tlo_type) {
141          if (tloPtr->root.objName == NULL)
142              return(EP_RB_RECOVER_INVALID_OBJNAME);
143      }
144      else if (objectLevel == RSTRPC_container_type) {
145          if (uroPtr->root.objName == NULL)
146              return(EP_RB_RECOVER_INVALID_OBJNAME);
147      }
148  })
149  return(EP_RB_RECOVER_INVALID_OBJNAME);
150  }

```

```

151  }
152  if (rcp->rc_backup_app != uroPtr->root.backupApp)
153  {
154      /* input object not from same app as current TLO */
155      return EP_RB_RECOVER_INVALIDOP;
156  }
157  else if (objectLevel == RSTRPC_leaf_type) {
158      return(EP_RB_RECOVER_INVALIDOP);
159  }
160  else {
161      return(EP_RB_RECOVER_INVALID_OBTYP);
162  }
163
164  /*
165  * If cookie is NULL of if it indicates DONE_COOKIE, return error.
166  */
167  if ((cookie == NULL) || (*cookie == DONE_COOKIE))
168  {
169      return(EP_RB_RECOVER_BAD_COOKIE);
170  }
171
172  /*
173  * If EDMNST_Initialize() has not been called, or for some reason
174  * the recover_context has not been allocated, then bail out
175  * right away!
176  */
177  if (rcp == NULL)
178  {
179      return(EP_RB_RECOVER_INVALIDOP);
180  }
181
182  /*
183  * If EDMNST_GetTopLevelObjects has not been called,
184  * then this operation
185  * is invalid.
186  */
187  if (rcp->rc_source_client_hostname == NULL)
188  {
189      return(EP_RB_RECOVER_INVALIDOP);
190  }
191
192  /*
193  * Set number of entries returned to be 0 in case we encounter an
194  * error before setting it appropriately.
195  */
196  *cntEntries = 0;
197
198  /*
199  * If the parent object is a top level object,
200  * call GetTLOContents();
201  * if parent object is a container (
202  *   directory), call RSTSL_GetDirContents
203  *   for network backups,
204  *   and RSTPL_GetNextLevelObjects for other backup
205  *   apps;
206  * else return error indicating parent object is not a parent.
207  */
208  if (objectLevel == RSTRPC_tlo_type) {
209      result = GetTLOContents( tloPtr,
210                              allowBF,
211                              maxEntries,

```

```

212 2      objects,
213 2      cntEntries,
214 2      cookie );
215 1      }
216 1      else
217 2      {
218 2          /* we already know its a container */
219 2      }
220 2      /* Top Level Object must have been established before this
221 2      func
222 2      * is called to list dir contents.
223 2      */
224 2      if (rcp->rc_top_level_object_name == NULL)
225 2      {
226 2          return EP_RB_RECOVER_INVALID;
227 2      }
228 2      if (NULL != rcp->currentPiptr)
229 2      {
230 2          result =
231 2              rcp->currentPiptr->piFuncArray[piFuncIndexGetNIO]
232 2              ( rcp,
233 2                uroPtr,
234 2                RSTRPC_container_type,
235 2                objects,
236 2                cookie,
237 2                maxEntries,
238 2                cntEntries,
239 2                allowBF );
240 2      }
241 2      else
242 2      {
243 2          result = RSTLL_GetDirContents( rcp,
244 2              uroPtr->root.objName,
245 2              allowBF,
246 2              maxEntries,
247 2              objects,
248 2              cntEntries,
249 2              cookie );
250 2      }
251 2      /* if successful, mark all objects with proper backup app: */
252 2      if (E_SUCCESS == result)
253 2      {
254 2          for ( urolist = *objects; urolist; urolist = urolist->next )
255 2          {
256 2              urolist->uro->root.backupApp = rcp->rc_backup_app;
257 2          }
258 2      }
259 2      return result;
260 2      }
261 2      /* RSTSL_GetRestorableObjects */

```

```

263 2      /*
264 2      * GetTLOCContents()
265 2      * Function Description:
266 2      *   Given the name of the top level object,
267 2      *   set up the restore_context
268 2      *   for it.
269 2      *
270 2      * Parameters:
271 2      *   tloObjPtr - (I) ptr to the work item restorableobject
272 2      *   allowBF - (I) flag indicating whether or not to include bad files
273 2      *   maxEntries - (I) max. # of entries that the preallocated buffer can
274 2      *   hold
275 2      *   objListPtr - (I) a pre-allocated pointer to return the list of
276 2      *   objects in
277 2      *   cntEntries - (I) ptr to buffer to receive number of entries returned
278 2      *   in objListPtr
279 2      *   cookie - (I) ptr to a long integer whose value is meaningful to
280 2      *   only the internals of the API
281 2      *
282 2      * Return code:
283 2      *   E_SUCCESS - success. (It is defined as 0 in e_errno.h)
284 2      */
285 2
286 2      static e_errno_ty
287 2      GetTLOCContents(
288 2          struct RSTRPC_top_level_obj *tloPtr,
289 2          const boolean_ty allowBF,
290 2          const long maxEntries,
291 2          struct RSTRPC_uro_list **objListPtr,
292 2          long *cntEntries,
293 2          long *cookie)
294 2      {
295 2          char *tloNameP = tloPtr->root.objName;
296 2          char *rcObjNameP;
297 2          e_errno_ty rc;
298 2          boolean_ty reset = FALSE;
299 2          int index;
300 2
301 2          /*
302 2          * The environment needs to be reset under the following
303 2          * situations:
304 2          * - It's the first time ever that a top level object is selected;
305 2          * - the caller wants to switch to a top level object different
306 2          *   from
307 2          * what the env is set up for currently;
308 2          *
309 2          * The following criteria are used to determine if the caller
310 2          * wants to switch to a different top level object:
311 2          * - The backup application in the restorableobject is
312 2          *   different from what's kept in the restore_context;
313 2          * - the object name specified in the restorableobject is
314 2          *   different from what's kept in the restore_context;
315 2          * - the object name in the restorableobject is the same
316 2          *   as what's in the restore_context, but the template or
317 2          *   the traitset used is different,
318 2          * or the client host is different;
319 2          */

```

```

320 1         if ( (0 != strcmp(
321 1             tloPtr->hostname, rcp->rc_source_client_hostname ) )
322 2             || (rcp->rc_backup_app != tloPtr->root.backupApp) )
323 2             {
324 1                 reset = TRUE; /* switch to a different application! */
325 1             }
326 1         else if ( (rcp->rc_top_level_object_name == NULL)
327 1             || (0 != strcmp(rcp->rc_top_level_object_name, tloNameP)) )
328 2             {
329 2                 reset = TRUE; /* switch to a different object */
330 1             }
331 1         else
332 2             {
333 2                 /* rc_top_level_object_name is already set and it's the same
334 2                 * as the input top level obj name, we need to further check
335 2                 * the template. If the template is the same, we must
336 2                 * then check the trailset usage. If any of these is
337 2                 * different between what's in the input restorableObject
338 2                 * and what's in the restore_context, we need to reset
339 2                 * the environment.
340 2                 */
341 2             }
342 2         if (rcp->rc_template_defaulted)
343 2             rctmpNameP = NULL;
344 2         else
345 2             rctmpNameP = rcp->rc_template_name;
346 2
347 2         if (NULL != tNameP)
348 2             {
349 3                 if (NULL == rctmpNameP)
350 3                     {
351 4                         reset = TRUE;
352 4                     }
353 3                 else if (strcmp(tNameP, rctmpNameP) /* diff templ */)
354 3                     {
355 4                         reset = TRUE;
356 4                     }
357 3                 else if (rcp->rc_saveset_thread != tloPtr->ssThread)
358 3                     {
359 4                         reset = TRUE;
360 4                     }
361 3                 }
362 2             }
363 2         else if (NULL != rctmpNameP)
364 3             {
365 3                 reset = TRUE;
366 2             }
367 1         }
368 1         if (reset)
369 1             {
370 2                 if (*cookie != INIT_COOKIE)
371 2                     {
372 3                         return(EP_RB_RECOVER_INVALID);
373 3                     }
374 2             }
375 2         /* if last tlo was other than a network backup object,
376 2         let app clean up: */
377 2         if (rcp->rc_top_level_object_name != NULL)
378 3             {
379 3                 if (NULL != rcp->currentPiptr)
380 4                     {
381 4                         rcp->currentPiptr->PFuncArray[PFuncIndexClearRC] ( rcp );

```

```

382 4         if (rc != E_SUCCESS)
383 5             {
384 5                 rbe_internal_error( rc,
385 5                     "plugin: %s error in
386 5                     clearRestoreContext call",
387 4                     rcp->currentPiptr->iddata )->name );
388 4             }
389 5         rcp->currentPiptr->appData = rcp->appData; /* save its data */
390 5         }
391 5         else
392 4             {
393 4                 /* for network backups, just clear the mark context: */
394 4                 reset_marks( rcp );
395 3             }
396 3         free( rcp->rc_top_level_object_name );
397 3         rcp->rc_top_level_object_name = NULL;
398 2         }
399 2         /*
400 2         * If we're switching to a new source host, we must free
401 2         * the space used for the previous source host name string
402 2         * before
403 2         * resetting it to the new.
404 2         */
405 2         if (0 != strcmp(
406 2             tloPtr->hostname, rcp->rc_source_client_hostname) )
407 3             {
408 3                 if (NULL != rcp->rc_source_client_hostname)
409 4                     {
410 4                         free(rcp->rc_source_client_hostname);
411 3                     }
412 3                 rcp->rc_source_client_hostname = esl_strdup(
413 3                     tloPtr->hostname );
414 3                 if (NULL == rcp->rc_source_client_hostname)
415 4                     {
416 4                         rec_api_log_csm(SUB_CSM_NOMEM, NULL);
417 3                     }
418 2                 return(EP_RB_RECOVER_NOMEM);
419 2             }
420 2         if (0 == (rcp->rc_backup_app = tloPtr->root.backupApp) )
421 3             {
422 3                 /* new app is network: */
423 3                 rcp->appData = NULL;
424 2                 rcp->currentPiptr = NULL;
425 2             }
426 3         else /* find app in plugin list,
427 3         set its appData and pluginData ptrs */
428 3             {
429 3                 for (index=1, rcp->currentPiptr = rcp->pilist;
430 3                     index < rcp->rc_backup_app &&
431 3                     rcp->currentPiptr->next ;
432 3                     index++)
433 3                     {
434 4                         rcp->currentPiptr = rcp->currentPiptr->next;
435 4                         if (index != rcp->rc_backup_app)
436 4                             { /* got null ptr too early */
437 4                                 rbe_internal_error( EP_RB_RECOVER_FATALERR,
438 4                                     "plug-in list corrupted"
439 4                                     );
440 4                                 rec_api_log_csm( SUB_CSM_INTERNAL_ERR, NULL );
441 4                                 return EP_RB_RECOVER_FATALERR;
442 4                             }
443 3                     }

```

```

440 3      rcp->appData = rcp->currentPiptr->appData;
441 2    }
442 2
443 2    /*
444 2     * If we're switching to a different tlo, we must:
445 2     * unlock any previous work items and free the workitem name
446 2     */
447 2     if (rcp->rc_workitem_name != NULL)
448 2     {
449 3         if (rcp->rc_have_wilock)
450 3         {
451 4             eb_unlock_object(
452 4                 EB_OBJECT_WORKITEM, rcp->rc_workitem_name,
453 4                 EB_UNLOCK_FREE_IT);
454 3             rcp->rc_have_wilock = 0;
455 3             free(rcp->rc_workitem_name);
456 3             rcp->rc_workitem_name = NULL;
457 2         }
458 2     }
459 2     if (rcp->rc_template_name != NULL)
460 2     {
461 3         free(rcp->rc_template_name);
462 2     }
463 2     if (tNameP != NULL)
464 2     {
465 3         rcp->rc_template_name = esl_strdup( tNameP );
466 3         rcp->rc_template_defaulted = FALSE;
467 3         if (NULL == rcp->rc_template_name)
468 3         {
469 4             rec_api_log_csm(SUB_CSM_NOMEM, NULL);
470 4             return(EP_RB_RECOVER_NOMEM);
471 4         }
472 3     }
473 2     else
474 2     {
475 3         rcp->rc_template_name = NULL;
476 3         rcp->rc_template_defaulted = TRUE;
477 3     }
478 2     rcp->rc_save_set_thread = tloPtr->ssThread;
479 2
480 2     /* Do this last,
481 2      * to make sure all strdup's succeed: else leave
482 2      * rc_top_level_object_name NULL */
483 2     rcp->rc_top_level_object_name = esl_strdup(tloNameP);
484 2     if (NULL == rcp->rc_top_level_object_name)
485 2     {
486 3         rec_api_log_csm(SUB_CSM_NOMEM, NULL);
487 3         return(EP_RB_RECOVER_NOMEM);
488 3     }
489 2
490 2     /*
491 2     * Check if restore is authorized to the specified client
492 2     * and by the specified user.
493 2     */
494 2     check_source_sys_admin(rcp);
495 2     if (0 != rcp->rc_backup_app)
496 2     {
497 3         rc =
498 3
499 3
500 3

```

```

501 3      rcp->currentPiptr->piFuncArray[piFuncIndexGetTLO]
502 2      {
503 2          rcp, tloPtr;
504 2      }
505 2      else
506 2      {
507 3          /* for network backups, init the workitem & mark context: */
508 3          rc = RSTLL_SetWorkitem( rcp, tloNameP );
509 3          if (rc == E_SUCCESS)
510 3          {
511 4              rc = alloc_plane_arrays( rcp );
512 4          }
513 3          if (rc != E_SUCCESS)
514 3          {
515 4              free( rcp->rc_top_level_object_name );
516 4              rcp->rc_top_level_object_name = NULL;
517 4              return(rc);
518 3          }
519 2          if (0 != rcp->rc_backup_app)
520 2          {
521 3              return rcp -> currentPiptr -> piFuncArray[piFuncIndexGetTLO]
522 3              { rcp,
523 3                  tloPtr,
524 3                  RSTRPC_tlo_type,
525 3                  objListPtr,
526 3                  cookie,
527 3                  maxEntries,
528 3                  objListPtr,
529 3                  cntEntries,
530 3                  allowBF );
531 2          }
532 2          /* for network backups retrieve content for the root directory. */
533 2          {
534 3              return RSTLL_GetDirContents( rcp,
535 3              /*",
536 3              allowBF,
537 3              maxEntries,
538 3              objListPtr,
539 3              cntEntries,
540 3              cookie );
541 3          }
542 2          /* GetTLOContents */
543 2      }

```


D6

EDMRST_GetMarkResults	5	(RSTmarkumm.c)
EDMRST_GetMarkedTotalSize...	11	(RSTmarkumm.c)
EDMRST_GetUnmarkResults	9	(RSTmarkumm.c)
EDMRST_MarkObject.....	2	(RSTmarkumm.c)
EDMRST_UnmarkObject	7	(RSTmarkumm.c)

```

2  /*****
3  **
4  ** File Name:   RSTmarkum.c
5  **
6  ** Copyright (c) 1998, 1999 by EMC Corporation.
7  **
8  ** Purpose:
9  **           This module contains the Restore API functions to
10 **           unmark objects for restore.
11 **
12 ** Table of Contents:
13 ** -----
14 ** API Functions:
15 **     EDMRST_MarkObject
16 **     EDMRST_GetMarkResults
17 **     EDMRST_UnmarkObject
18 **     EDMRST_GetUnmarkResults
19 **     EDMRST_GetMarkedTotalSize
20 **
21 **     Internal Functions:
22 **
23 ** Compile-Time Options:
24 **
25 **
26 ** NOTE: Part of this module is adapted from:
27 ** server/libs/recover/grandfathered/cmd/markumark.c
28 ** It contains mainly support routines needed by the mark and unmark
29 ** API functions.
30 **
31 *****/

32
33 /* The following provides an RCS id in the binary that can be located
34 ** with the what(1) utility. The intent is to keep this short.
35 **/
36
37 #ifndef lint
38 static char RCS_id [] = "$RCSfile$"
39                          "$Revision$"
40                          "$Date$";
41 #endif
42
43 /*
44 ** Feature test switches.
45 ** Standard defines required to turn on OS features go here.
46 **
47 ** The following is required for code that uses POSIX APIs.
48 ** Remove for non-POSIX, non-portable code.
49 **/
50
51 #define _POSIX_SOURCE 1
52
53 /*
54 ** System headers.
55 **/
56
57 /*
58 ** Epoch headers.
59 **/
60
61
62
63

```

```

64 #include <eb/eb_port.h>
65 #include <eb/rb_log.h>
66
67 /*
68 ** Local headers
69 **/
70 #include <RSTinterns.h>
71 #include <RSTsup_csm.h>
72
73 /*
74 ** #defines, structures, typedefs local to this source file
75 **/
76
77
78 /*
79 ** External declarations
80 **/
81
82
83
84
85 NEW_SRC_FILE();
86
87 /*
88 ** Local function prototypes
89 **/
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121

```

```
122         time_t          time,
123         boolean_ty      allowBadFiles,
124         boolean_ty      descend )
125     {
126         RE_mark_object_result *rpc_result;
127         RE_mark_object_args   rpc_args;
128         RSTRPC_restorable_obj_root result = E_SUCCESS ;
129         eerrno_ty
130
131         rbe_log_debug_sub( 0, "EDMRST_MarkObject called" );
132
133         /* validate args first: */
134         if (thisobject==NULL || svrHdl==NULL )
135             return( EP_RB_RECOVER_BAD_ARGS );
136
137         /* validate input object type as RESTORABLE_OBJECT */
138         temp_obj = (restorableobject *)thisobject->ipcObjPtr;
139         if ( NULL == temp_obj
140             || RESTORABLE_OBJECT !=
141                 (
142                     restorableobject *)thisobject->restoreObjType )
143             return EP_RB_RECOVER_INVALID_OBJTYPE;
144
145         /* validate input object type as NOT top level */
146         if ( (temp_obj->objLevel != RSTRPC_leaf_type)
147             && (temp_obj->objLevel != RSTRPC_container_type) )
148             {
149                 if (temp_obj->objLevel != RSTRPC_tlo_type)
150                     return( EP_RB_RECOVER_INVALID_OBJTYPE );
151                 else
152                     return( EP_RB_RECOVER_INVALIDOP );
153             }
154
155         /* Prepare input argument structure for RPC: */
156         rpc_args.thisobj = (RSTRPC_user_restorable_object *)temp_obj;
157         rpc_args.allowBadFiles = allowBadFiles;
158         rpc_args.descend = descend;
159         rpc_args.backupTime = time;
160         set_rpc_obj( re_mark_object, &rpc_args.RPCobjID );
161
162         rpc_result = re_mark_object_1( &rpc_args, svrHdl );
163
164         if (!rpc_result) {
165             result = EP_RB_RECOVER_RPC_FAIL;
166             rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL );
167         }
168         else {
169             result = rpc_result->status;
170             /* release RPC result struct: */
171             xdr_free( xdr_RE_mark_object_result, (
172                 char *)rpc_result);
173             return( result );
174         }
175         /* end of EDMRST_MarkObject ( ) */
```

File Oct 10 12:11:25 2008	EDMRST_GetMarkResults	Page 5 of 12
177	*****	
178	EDMRST_GetMarkResults()	
179	* This function tests for completion and retrieves the results	
180	* of the	
181	* previously started mark operation.	
182	* Parameters:	
183	* svrHdl (I) - A pointer to this user's client handle for the	
184	* Restore Engine (server) connection.	
185	* interrupt (I) - Requests cancellation of the mark (if TRUE)	
186	* WARNING: If the operation is aborted,	
187	* the mark will be	
188	* left in an unknown state. That is, any one of the	
189	* descendants of the marked object may be marked	
190	* or not.	
191	* It is up to the caller to determine how to	
192	* proceed	
193	* afterwards.	
194	* BadFilesCount (
195	* 0) -- returns the file count of bitfiles marked with BADDATA	
196	* PermdenyFilesCount (
197	* 0) -- returns the file count with permission denied	
198	* bitfiles that were not marked.	
199	* fileMarked (
200	* 0) -- return the total files marked after this mark occurred.	
201	* dirMarked (
202	* 0) -- return the total directories marked after this mark	
203	* otherMarked (
204	* 0) -- return the total "other" files marked after this mark.	
205	*****	
206	eeerrno_ty	
207	EDMRST_GetMarkResults (
208	serverHandle	
209	svrHdl,	
210	boolean_ty	
211	interrupt,	
212	u_long	
213	*BadFilesCount,	
214	*PermdenyFilesCount,	
215	*fileMarked,	
216	*dirMarked,	
217	*otherMarked)	
218	{	
219	RE_get_mark_results_result	
220	rpc_args;	
221	result = E_SUCCESS ;	
222	eeerrno_ty	
223	rybe_log_debug_sub(0, "EDMRST_GetMarkResults called");	
224	/* validate args first: */	
225	if (svrHdl==NULL BadFilesCount==NULL	
226	fileMarked==NULL PermdenyFilesCount==NULL	
227	dirMarked==NULL otherMarked==NULL	
228)	
229	return(EP_RB_RECOVER_BAD_ARGS);	
230	rpc_args.interrupt = interrupt;	
231	set_rpc_obj(re_get_mark_results, &rpc_args.RPCobjID);	
232	rpc_result = re_get_mark_results_1(&rpc_args, svrHdl);	
233	if (!rpc_result) {	
234	result = EP_RB_RECOVER_RPC_FAIL;	
235	}	
236	return result;	
237	}	
238	}	
239	}	
240	}	
241	}	
242	}	
243	}	
244	}	
245	}	
246	}	
247	}	
248	}	
249	}	
250	}	
251	}	

Fri Oct 10 12:11:25 2008		EDMRST_GetMarkResults	Page 6 of 12
233	2	rec_api_log_csm(SUB_CSM_RPC_FAIL, NULL);	
234	1	}	
235	2	else {	
236	2	/* move results to caller's area, if successful: */	
237	2	result = rpc_result->status;	
238	3	if (result == E_SUCCESS)	
239	3	{	
240	3	*BadFilesCount = rpc_result->badFileCount;	
241	3	*PermDenyFilesCount =	
242	3	rpc_result->permDenyFileCount;	
243	3	*fileMarked = rpc_result->fileMarkCount;	
244	2	*otherMarked = rpc_result->otherMarkCount;	
245		}	
246	2	/* release RPC result struct: */	
247	2	xdr_free(xdr_RE_get_mark_results_result, (
248	1	char *)rpc_result);	
249		}	
250	1	return(result);	
251		}	
Fri Oct 10 12:11:25 2008		RSTmarkum.c 6	Page 6 of 12

```

253 /*****
254  * UnmarkObject() and GetUnmarkResults()
255  *
256  * UnmarkObject operates like MarkObject,
257  *   in that it is supported through
258  *   two API calls -- UnmarkObject and GetUnmarkResults.
259  *   Unmark starts an
260  *   asynchronous operation in the Restore Engine to perform the
261  *   unmarking,
262  *   and returns.
263  *   GetUnmarkResults is called to test for completion of the unmark
264  *   operation,
265  *   and receive results when it is done.
266  *   It can also be used to interrupt
267  *   the unmark operation.
268  *
269  * UnmarkObject Parameters:
270  *
271  *   svrHdl      (I) - A pointer to this user's client handle for the
272  *   Restore Engine (server) connection.
273  *   thisObject  (I) - The restoral object;
274  *   can be a leaf object (e.g. a
275  *   file), or a container object (
276  *   e.g., a directory).
277  *
278  *   backupTime  (I) - (
279  *   optional) the backup time to perform the unmark on --
280  *   if not specified,
281  *   uses currently selected backup; if
282  *   specified,
283  *   leaves selected backup time unchanged
284  *
285  *   BadFilesOnly (I) - allows unmarking ONLY of files of state BADDATA.
286  *
287  *   descend      (I) - Should unmark operation descend to operate on the
288  *   content of container objects.
289  *
290  *   ****
291  *   eerrno_ty EDMRST_UnmarkObject( serverHandle svrHdl,
292  *   restoreableObjectPtr thisObject,
293  *   time_t backupTime,
294  *   boolean_ty BadFilesOnly,
295  *   boolean_ty descend )
296  *
297  *   RE_mark_object_result *rpc_result;
298  *   RE_unmark_object_args
299  *   RSTRPC_restorable_obj_root *temp_robj;
300  *   eerrno_ty result = E_SUCCESS ;
301  *
302  *   rbe_log_debug_sub( 0, "EDMRST_UnmarkObject called" );
303  *
304  *   /* validate args first: */
305  *   if (thisObject==NULL || svrHdl==NULL )
306  *       return ( EP_RB_RECOVER_BAD_ARGS );
307  *
308  *   /* validate input object type as RESTORABLE_OBJECT */
309  *   temp_robj = ((restorableObject *)thisObject)->rpcobjPtr;
310  *   if (NULL == temp_robj || RESTORABLE_OBJECT !=
311  *       (
312  *           restorableObject *)thisObject->restoreobjType )
313  *       return EP_RB_RECOVER_INVALID_OBTYP;
314  *
315  *   /* validate input object type as NOT top level */
316  *   if ( (temp_robj->objLevel != RSTRPC_leaf_type)
317  *       && (temp_robj->objLevel != RSTRPC_container_type) )
318  *       return( result );
319  *
320  *   RSTmarkum.c 7
321  */

```

```

305 2 {
306 2     if (temp_robj->objLevel != RSTRPC_tlo_type)
307 2         return( EP_RB_RECOVER_INVALID_OBTYP );
308 2     else
309 2         return( EP_RB_RECOVER_INVALID );
310 1 }
311 1
312 1 /* Prepare input argument structure for RPC: */
313 1 rpc_args.thisObj = (RSTRPC_user_restorable_object *)temp_robj;
314 1 rpc_args.badFilesOnly = BadFilesOnly;
315 1 rpc_args.descend = descend;
316 1 rpc_args.backupTime = backupTime;
317 1 set_rpc_obj( re_unmark_object, &rpc_args.RPCobjID );
318 1
319 1 rpc_result = re_unmark_object_1( &rpc_args, svrHdl );
320 1
321 2 if (!rpc_result) {
322 2     result = EP_RB_RECOVER_RPC_FAIL;
323 2     rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL );
324 1 }
325 2 else {
326 2     result = rpc_result->status;
327 2     /* release RPC result struct: */
328 2     xdr_free( xdr_RB_mark_object_result, (
329 1         char *)rpc_result );
330 1 }
331 1 return( result );
332 1 }

```

```
333  / * end of EDMRST_UnmarkObject ( ) */
335  /*****
336  *
337  * GetUnmarkResults Parameters:
338  *
339  * svrHdl      (I) - A pointer to this user's client handle for the
340  *               Restore Engine (server) connection.
341  * interrupt   (I) - requests cancellation of the unmark (if TRUE)
342  *               WARNING: If the operation is aborted,
343  *               the unmark will
344  *               be left in an unknown state. That is,
345  *               any one of the
346  *               descendants of the unmarked object may be
347  *               marked or
348  *               not.
349  *               It is up to the caller to determine how to
350  *               proceed afterwards.
351  * BadFilesCount (O) - returns the file count with BADDATA.
352  * fileMarked   (
353  *               O) - return the total files marked after this mark
354  *               occurred.
355  * dirMarked    (
356  *               O) - return the total directories marked after this mark
357  *               occurred.
358  * otherMarked  (
359  *               O) - return the total "other" files marked after this mark.
360  *
361  * *****/
362  eerrno_t EDMRST_GetUnmarkResults( serverHandle svrHdl,
363  boolean_t interrupt,
364  u_long *BadFilesCount,
365  u_long *fileMarked,
366  u_long *dirMarked,
367  u_long *otherMarked )
368  {
369  RE_get_unmark_results_result *rpc_result;
370  RE_get_mark_results_args    result = E_SUCCESS ;
371  eerrno_t
372  rbe_log_debug_sub( 0, "EDMRST_GetUnmarkResults called" );
373
374  /* validate args first: */
375  if ( svrHdl==NULL || BadFilesCount==NULL || fileMarked==NULL
376  || dirMarked==NULL || otherMarked==NULL
377  )
378  return( EP_RB_RECOVER_BAD_ARGS );
379
380  rpc_args.interrupt = interrupt;
381  set_rpc_obj( re_get_unmark_results, &rpc_args, RPObjID );
382  rpc_result = re_get_unmark_results_1( &rpc_args, svrHdl );
383
384  if ( !rpc_result )
385  {
386  result = EP_RB_RECOVER_RPC_FAIL;
387  rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL );
388  }
389  else { /* move results to caller's area, if successful: */
390  if (result == E_SUCCESS)
391  {
392  *BadFilesCount = rpc_result->badfileCount;
393  *fileMarked = rpc_result->fileMarkCount;
394  *dirMarked = rpc_result->dirMarkCount;
395  *otherMarked = rpc_result->otherMarkCount;
396  }
```

```
392 2  / * release RPC result struct: */
393 2  xdr_free( xdr_RE_get_unmark_results_result, (
394 1  char *)rpc_result );
395 1  }
396 1  return( result );
397  }
```

```
399 /*****
400  * EDMRST_GetMarkedTotalSize ( )
401  *
402  * This function is provided to allow retrieval of the
403  * Total size of the marked files.
404  *
405  * size is the sum-of-the-length the marked files and is one
406  * measure of size. This is an approximation.
407  *****/
408
409 eerrno_ty
410 EDMRST_GetMarkedTotalSize( serverhandle svrHdl,
411                             u_hyper *totalSize )
412
413 {
414     RE_get_marked_total_size_result *rpc_result;
415     RE_null_args
416     eerrno_ty
417     rbe_log_debug_sub( 0, "EDMRST_GetMarkedTotalSize called" );
418
419     /* validate args first: */
420     if ( svrHdl==NULL || totalSize==NULL )
421         return( EP_RB_RECOVER_BAD_ARGS );
422
423     set_rpc_obj( re_get_marked_total_size, &rpc_args.RpcobjID );
424     rpc_result = re_get_marked_total_size_1( &rpc_args, svrHdl );
425
426     if (!rpc_result) {
427         result = EP_RB_RECOVER_RPC_FAIL;
428         rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL );
429     }
430     else { /* move results to caller's area, if successful: */
431         result = rpc_result->status;
432         if (result == E_SUCCESS)
433             {
434                 totalSize->high = rpc_result->total.high;
435                 totalSize->low = rpc_result->total.low;
436             }
437
438         /* release RPC result struct: */
439         xdr_free( xdr_RE_get_marked_total_size_result,
440                  (char *)rpc_result );
441
442     }
443     return( result );
444 }
445
```